CESAR SCHOOL

JADSON DE ARAÚJO ALMEIDA

REVISÃO SISTEMÁTICA SOBRE A RELEVÂNCIA DOS PADRÕES DE PROJETO GOF PARA O DESENVOLVIMENTO DE JOGOS DIGITAIS MODERNOS

RECIFE
2018

JADSON DE ARAÚJO ALMEIDA

REVISÃO SISTEMÁTICA SOBRE A RELEVÂNCIA DOS PADRÕES DE PROJETO GOF PARA O DESENVOLVIMENTO DE JOGOS DIGITAIS MODERNOS

Dissertação apresentada ao programa de Mestrado em Engenharia de Software do Centro de Estudos e Sistemas Avançados do Recife – C.E.S.A.R, como requisito para a obtenção do título de Mestre em Engenharia de Software.

Orientação: Prof. MSc. Felipe Silveira Mello Borgiani

Dados Internacionais de Catalogação na Publicação (CIP)
Centro de Estudos e Sistemas Avançados do Recife - CESAR School

A447r Almeida, Jadson de Araújo

Revisão sistemática sobre a relevância dos padrões de projeto *GOF* para o desenvolvimento de jogos digitais modernos. / Jadson de Araújo Almeida. – Recife: O autor 2018.

98 p.: il.

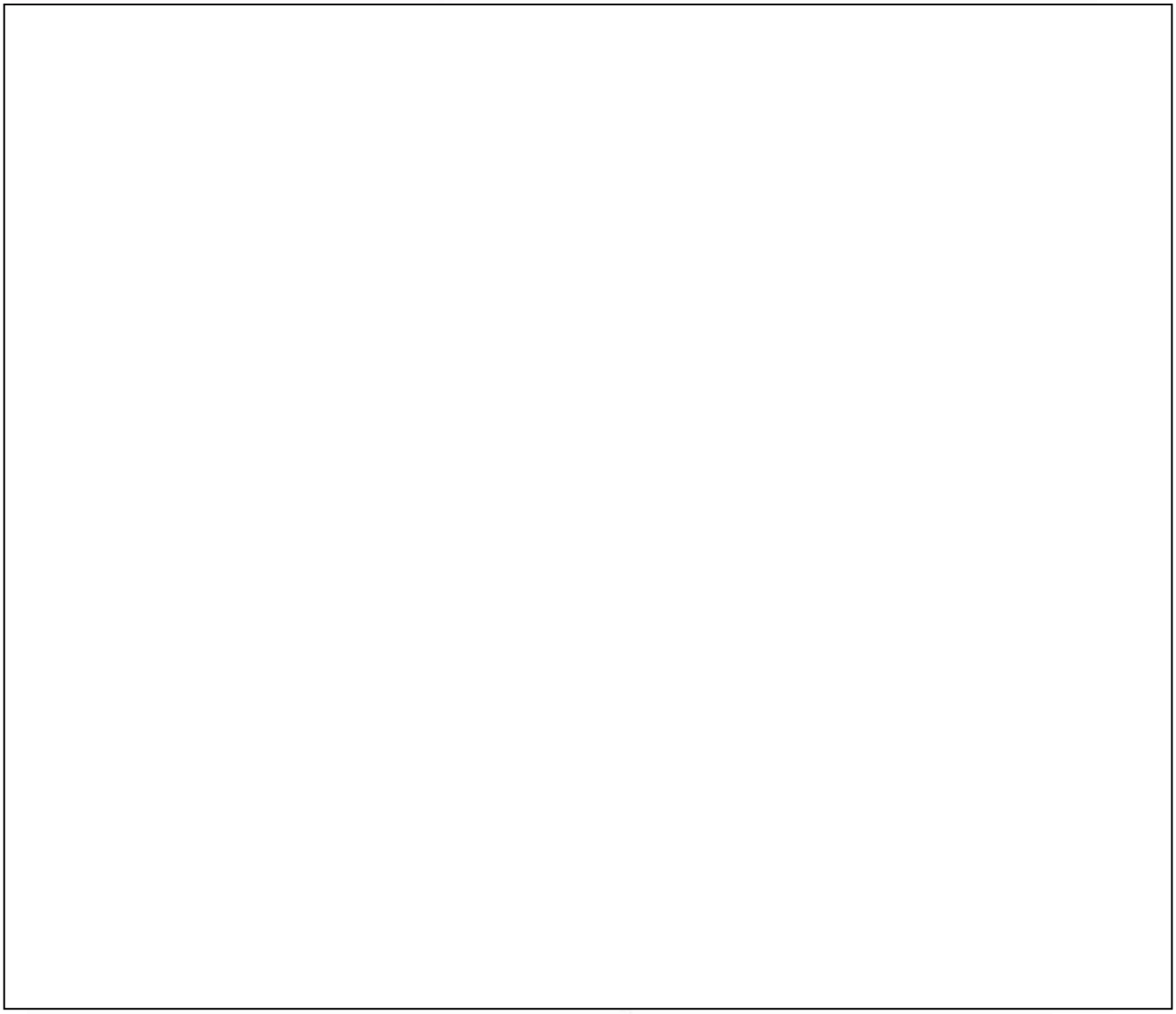
Dissertação (Mestrado) - Centro de Estudos e Sistemas Avançados do Recife - CESAR School. Programa de Pós-Graduação em Engenharia de Software.

Orientação: Prof. Felipe Silveira Mello Borgiani

1. Jogos Digitais Modernos. 2. Padrões de Projetos. 3. Desenvolvimento de Jogos. 4. *GOF.* I. Borgiani, Felipe Silveira Mello (Orientador). II. Título.

CDD 794

Dissertação de Mestrado apresentada por Jadson de Araújo Almeida ao Programa de Mestrado Profissional em Engenharia de Software do CESAR (Centro de Estudos e Sistemas Avançados do Recife) intitulada "Revisão Sistemática Sobre a Relevância dos Padrões de Projeto Gof Para o Desenvolvimento de Jogos Digitais Modernos", orientada por: Felipe Silveira Mello Borgiani. A dissertação foi aprovada pela Banca Avaliadora formada por: Felipe Silveira Mello Borgiani – CESAR School, Walquíria Castelo Branco Lins – CESAR School e Ewerton Menezes de Mendonça - Unifavip - Departamento de Engenharia.



Dedicatória

Dedico este trabalho aos meus pais, sra. Ady Gomes de Araújo Almeida e sr. Severino de Almeida Calado Filho.

Agradecimentos

Nada além de justo é agradecer àqueles que, além da vida, me criaram com dedicação e afeto. Aos meus amados pais, toda a gratidão e honra que me é possível. Sem o apoio deles, nada disso existiria.

Agradeço ao parceiro de conversas sobre jogos, irmão Jonatas Almeida, e ao apoio emocional dado pelo irmão Jefferson Almeida e sua esposa – e minha amiga – Fernanda Felix.

Não poderia deixar de agradecer os tutores Rodrigo Lopes, César França, Ivan Samuel, Ewerton Mendonça, Gustavo Alexandre e meu orientador: Felipe Borgiani. Muito do que aprendi e consegui eu devo ao apoio e exemplo de vocês.

Também, meu eterno reconhecimento pelos amigos Bira Nunes, José Marcos e Lucas Ximenes pelas horas de RPG abdicadas; a Saulo Gonçalves por acompanhar e incentivar minha trajetória acadêmica desde o início; a Fhelipe Freitas pelos inúmeros áudios e trocas de informações sobre nossos trabalhos; a Fillype Augusto pelas ~~1000 horas de PUBG jogadas~~ horas de ludicidade durante esse período, e, a Juliana Barbosa pelo suporte e puxões de orelha.

Epígrafe

“Se te apetece esforçar, esforça-te; se te apetece repousar, repousa; se te apetece fugir, fuja; se te apetece resistir, resista. Mas, saiba bem o que te apetece e não recue ante nenhum pretexto, porque o universo se organizará para te dissuadir.”

(Interpretação de Clovis de Barros Filho para parte da obra “A Vontade de Poder” de Friedrich Nietzsche)

Resumo

No cenário atual de desenvolvimento de jogos digitais, é crescente o número de ferramentas que vão desde frameworks até a algoritmos otimizados. Jogos com investimentos de milhões de dólares e anos de desenvolvimento são cada vez mais comuns, agregando grande valor a reusabilidade de software. Famosos por essa propriedade, os padrões de projetos GoF são abordados também em trabalhos acadêmicos sobre jogos digitais. Assim, este trabalho intenciona analisar a pertinência do uso dos padrões de projeto GoF no desenvolvimento de jogos digitais modernos, levantando questionamentos sobre o tema através de uma revisão sistemática da literatura. Ao percorrer os 23 padrões de projetos GoF, o trabalho extrai da revisão sistemática, para cada padrão, seus atributos e métricas de qualidade de software, uma implementação solucionando uma funcionalidade de jogo digital, e, propõe uma equivalência entre esta funcionalidade e uma funcionalidade de um jogo digital moderno, afim de inferir a pertinência deste padrão no atual cenário de desenvolvimento de jogos digitais. Por fim, este trabalho aponta possíveis pesquisas a serem derivadas desses resultados.

Palavras-chave

Padrão de Projetos GoF. Desenvolvimento de Jogos. Atributos de qualidade de software.

Abstract

In the current scenario of game development, the number of tools like frameworks and optimized algorithms is increasing. Games with investments of millions of dollars and years of development are increasingly common, adding great value to software reusability. Famous for this property, the GoF design patterns are also treated in academic work about digital games. This work intends to analyze the pertinence of the use of GoF design patterns in the development of modern digital games, making questions about the theme through a systematic review of the literature. Going through the 23 patterns of GoF projects, the work draws from the systematic review, for each design pattern, its software quality attributes and metrics, an implementation solving digital game functionalities, and proposes an equivalence between this functionality and a functionality of a modern digital game, in order to infer the pertinence of this pattern in the current scenario of game development. Finally, this work points out possible research to be derived from these results.

Key-words

GoF Design Patterns. Game Development. Software quality attributes.

Lista de ilustrações

Diagrama 4-1 – Estrutura base do padrão de projetos *Abstract Factory*. 19
Diagrama 4-2 – Estrutura base do padrão de projetos *Builder*. 21
Diagrama 4-3 – Street of Rage sem o padrão de projetos *Builder*. 22
Diagrama 4-4 – Street of Rage com o padrão de projetos *Builder*. 23
Diagrama 4-5 – Estrutura base do padrão de projetos *Factory Method*. 24
Diagrama 4-6 – RPG com os padrões de projetos *Command*, *Factory Method* e *State*. 25
Diagrama 4-7 – Estrutura base do padrão de projetos *Prototype*. 26
Diagrama 4-8 – Criação de monstros com o padrão *Prototype*. 27
Diagrama 4-9 – Estrutura base do padrão de projetos *Singleton*. 29
Diagrama 4-10 – Versão do Reversi com o padrão de projetos *Singleton*. 30
Diagrama 4-11 – Estrutura base do padrão de projetos *Adapter*. 32
Diagrama 4-12 – *Adapter* na arquitetura MVC. 33
Diagrama 4-13 – *Adapter* convertendo *inputs*. 34
Diagrama 4-14 – Estrutura base do padrão de projetos *Bridge*. 34
Diagrama 4-15 – *Bridge* e representações. 36
Diagrama 4-16 – Estrutura base do padrão de projetos *Composite*. 37
Diagrama 4-17 – *Composite* no jogo Banco Imobiliário. 38
Diagrama 4-18 – Estrutura base do padrão de projetos *Decorator*. 40
Diagrama 4-19 – Composição de equipamentos sem o padrão de projetos *Decorator*. 41
Diagrama 4-20 – Composição de equipamentos com o padrão de projetos *Decorator*. 42
Diagrama 4-21 – Estrutura base do padrão de projetos *Façade*. 43
Diagrama 4-22 – *Façade* e desacoplamento. 44
Diagrama 4-23 – Estrutura base do padrão de projetos *Flyweight*. 45
Diagrama 4-24 – Estrutura base do padrão de projetos *Proxy*. 48
Diagrama 4-25 – Instanciação de cenários sem o padrão *Proxy*. 49
Diagrama 4-26 – Instanciação de cenários com o padrão *Proxy*. 50
Diagrama 4-27 – Estrutura base do padrão de projetos Chain of *Responsibility*. 52
Diagrama 4-28 – Tratamento de navegadores sem o padrão *Chain of Responsibility*. 53

Diagrama 4-29 – Tratamento de navegadores com o padrão *Chain of Responsibility*. 54
Diagrama 4-30 – Estrutura base do padrão de projetos *Composite*..... 55
Diagrama 4-31 – Estrutura base do padrão de projetos *Interpreter*. 58
Diagrama 4-32 – Estrutura base do padrão de projetos *Iterator*. 60
Diagrama 4-33 – Evolução de personagens sem o padrão *Iterator*. 61
Diagrama 4-34 – Evolução de personagens com o padrão *Iterator*. 62
Diagrama 4-35 – Estrutura base do padrão de projetos *Mediator*. 63
Diagrama 4-36 – Controle de colisões com *Mediator*, *Observer* e *Visitor*. 64
Diagrama 4-37 – Estrutura base do padrão de projetos *Memento*. 65
Diagrama 4-38 – Reversi com o padrão *Memento*. 66
Diagrama 4-39 – Estrutura base do padrão de projetos *Observer*. 68
Diagrama 4-40 – Gerenciador jogadores de futebol com o padrão *Observer*. 69
Diagrama 4-41 – Estrutura base do padrão de projetos *State*. 70
Diagrama 4-42 – Estados de jogo sem o padrão *State*. 71
Diagrama 4-43 – Estados de jogo com o padrão *State*. 72
Diagrama 4-44 – Estrutura base do padrão de projetos *Strategy*. 73
Diagrama 4-45 – Estrutura base do padrão de projetos *Template Method*. 77
Diagrama 4-46 – Motor de turnos com o padrão *Template Method*. 78
Diagrama 4-47 – Estrutura base do padrão de projetos *Visitor*. 79
Figura 2-1 – Quatro elementos dos jogos 8
Figura 4-1 – Árvore de habilidades do jogo *SpellForce* 3. 39
Figura 4-2 – Gerador de árvores sem *Flyweight*. 46
Figura 4-3 – Gerador de árvores com *Flyweight*. 47
Figura 4-4 – Controle do jogador sem o padrão *Command*. 56
Figura 4-5 – Controle do jogador com o padrão *Command*. 56
Figura 4-6 – Opções de controle do jogo Player's Unknows Battlegrounds. 57
Figura 4-7 – Campo de batalha do jogo SpellForce 3. 65
Figura 4-8 – Sprays de M416 em *Player's Unknows Battlegrounds*. 75
Figura 4-9 – Sprays de Vector em *Player's Unknows Battlegrounds*. 75
Figura 4-10 – Comportamentos do jogo *EEClone*. 80
Figura 4-11 – Uso do padrão *Visitor* no *EEClone*..... 81

Lista de tabelas

Tabela 1 – Relação entre propriedades dos jogos 6
Tabela 2 – Quantitativo de trabalhos inclusos e exclusos 14
Tabela 3 – Padrões de projetos GoF e Implementações em Jogos Digitais. 15
Tabela 4 – Trabalhos inclusos sobre padrões criacionais 18
Tabela 5 – Trabalhos inclusos sobre padrões estruturais 31
Tabela 6 – Trabalhos inclusos sobre padrões comportamentais 51

ABREVIATURAS

Sigla	Significado
GoF	Gang of Four, como ficou conhecido o quarteto de autores que escreveram o livro “Soluções Reutilizáveis de Software Orientado a Objetos”.
Hype	Adjetivo do que está em evidência no momento.
Jogo digital moderno	Jogos digitais publicados a partir de 2017 na plataforma Steam.
Padrão de projetos GoF	Padrão de Projeto descrito no livro “Soluções Reutilizáveis de Software Orientado a Objetos”.

SUMÁRIO

1 INTRODUÇÃO 1

1.1 TEMA 1

1.2 JUSTIFICATIVA 1

1.2.1 Objetivo geral 2

1.2.2 Objetivos específicos 3

1.3 ESTRUTURA DA DISSERTAÇÃO 3

2 FUNDAMENTAÇÃO TEÓRICA 5

2.1 REVISÃO SISTEMÁTICA DA LITERATURA 5

2.2 JOGOS DIGITAIS 6

2.3 ATRIBUTOS E MÉTRICAS DE QUALIDADE DE SOFTWARE 8

3 METODOLOGIA DE PESQUISA 11

3.1 PROTOCOLO DE PESQUISA 11

3.2 PERGUNTA DE PESQUISA 11

3.2.1 Pergunta específicas de pesquisa 11

3.3 ESTRATEGIA DE BUSCA 12

3.4 CRITÉRIOS DE INCLUSÃO E EXCLUSÃO 12

3.5 FONTES DE BUSCA 12

3.6 TERMOS DE BUSCA 13

3.7 STRING DE BUSCA 13

3.8 RESULTADOS 14

3.9 SELEÇÃO DOS RESULTADOS 14

3.10 EXTRAÇÃO DE DADOS 15

3.11 SÍNTESE DE DADOS 15

3.11.1 Síntese dos padrões de projetos gof 15

3.11.2 Síntese dos atributos e métricas de qualidade de software 16

4 DESENVOLVIMENTO E RESULTADOS 18

4.1 PADRÕES CRIACIONAIS 18

4.1.1 Abstract Factory 18

4.1.2 *Builder* 21

4.1.3 *Factory Method* 24

4.1.4 *Prototype* 26

4.1.5 *Singleton* 29

4.2 PADRÕES ESTRUTURAIS 31

4.2.1 *Adapter* 31

4.2.2 Bridge 34

4.2.3 ***Composite*** 36
4.2.4 ***Decorator*** 39
4.2.5 ***Façade*** 43
4.2.6 ***Flyweight*** 45
4.2.7 ***Proxy*** 48
4.3 PADRÕES COMPORTAMENTAIS 51
4.3.1 ***Chain of Responsibility*** 51
4.3.2 ***Command*** 54
4.3.3 ***Interpreter*** 57
4.3.4 ***Iterator*** 60
4.3.5 ***Mediator*** 62
4.3.6 ***Memento*** 65
4.3.7 ***Observer*** 67
4.3.8 ***State*** 70
4.3.9 ***Strategy*** 73
4.3.10 ***Template Method*** 76
4.3.11 ***Visitor*** 79
4.4 ANÁLISE DAS EVIDÊNCIAS 82
5 CONCLUSÃO 84
5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS 84
REFERÊNCIAS 86
APÊNDICE A – TRABALHOS INCLUÍDOS 91
APÊNDICE B – TRABALHOS NÃO OBTIDOS 92
APÊNDICE C – TRABALHOS QUE APRESENTAM SOLUÇÕES PARA FUNCIONALIDADES DE JOGOS DIGITAIS ATRAVÉS DO USO DE PADRÕES DE PROJETOS GOF CRIACIONAIS 93
APÊNDICE D – TRABALHOS QUE APRESENTAM SOLUÇÕES PARA FUNCIONALIDADES DE JOGOS DIGITAIS ATRAVÉS DO USO DE PADRÕES DE PROJETOS GOF ESTRUTURAIS 95
APÊNDICE E – TRABALHOS QUE APRESENTAM SOLUÇÕES PARA FUNCIONALIDADES DE JOGOS DIGITAIS ATRAVÉS DO USO DE PADRÕES DE PROJETOS GOF COMPORTAMENTAIS 96

1 INTRODUÇÃO

Esse capítulo inicia o contexto da pesquisa, a justificativa e objetivos da mesma e apresenta a estrutura deste trabalho.

1.1 TEMA

Na história dos jogos digitais, há discussões sobre qual foi o primeiro jogo digital publicado. Todavia, sabe-se que um dos primeiros jogos digitais de grande sucesso foi lançado oficialmente em 1962, o Spacewar! (LONNQUIST, 2011). Os jogos digitais, então, passaram a ser produzidos em grande escala, abordando diversos gêneros, tecnologias e plataformas. Segundo Mcdonald (2017) do portal de pesquisas Newzoo, a receita mundial dos jogos digitais ao longo de 2017 deve alcançar os 108,9 bilhões de dólares, com projeção de 128,5 bilhões para 2020. A receita chega a superar a soma da indústria musical e de cinema (STATISTA, 2016).

Ao tempo em que a tecnologia concedeu ferramentas como *engines*, APIs e frameworks para facilitar o desenvolvimento dos jogos digitais na área da engenharia de software, os jogos também evoluíram sua complexidade, englobando conexões em rede, motores gráficos, subsistemas inteligentes entre outros.

Além de softwares auxiliares, diversas facilidades foram criadas na engenharia de software para a simplificação dos algoritmos. Em especial, o livro Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos (GAMMA et al., 2007) que, segundo o autor, “descreve soluções simples para problemas específicos no projeto de software orientado a objetos. (...) captura soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo”. Os padrões de projeto têm enfoque na reusabilidade do código produzido e têm atraído a atenção dos pesquisadores, uma vez que são considerados uma parte respeitável da pesquisa e prática na engenharia de software (AMPATZOGLOU, CHARALAMPIDOU e STAMELOS, 2013).

1.2 JUSTIFICATIVA

Com a evolução dos jogos digitais, apesar das ferramentas e demais alternativas para facilitar o desenvolvimento dos jogos, o avanço tecnológico e o

consequente aumento da demanda do mercado pela evolução dos jogos, muitas características surgiram enquanto outras escalaram em alto nível. Assim, o desenvolvimento de jogos digitais continua a ser um desafio. No âmbito da programação, a complexidade dos algoritmos vai desde o tratamento na produção gráfica de última geração, produzindo um visual cada vez mais realista, chegando à meta de por vezes confundir os usuários sobre o que é virtual ou real. Os padrões de projeto GoF trazem soluções reutilizáveis à imensa quantidade de código gerado nos jogos digitais. O código de um servidor do jogo *Lineage 2: Freya* (TOTOV et al., 2004) escrito na linguagem Java chega a ter mais de mil e quinhentas classes. A reusabilidade passa a ser, em casos assim, um atributo de extrema importância. Além da reusabilidade, os autores Ampatzoglou, Charalampidou e Stamelos (2013) levantam uma lista de outros possíveis atributos presentes nos padrões de projeto GoF, como usabilidade, manutenibilidade, adaptabilidade entre outros.

A literatura traz trabalhos com aplicações de padrões GoF em contextos de jogos digitais, solucionando funcionalidades desses. Entretanto, alguns trabalhos utilizam jogos com mais de uma década de publicação, como *Altered Beast* e *Super Volleyball* citados por Roberto Figueiredo (2014), e, outros mencionam funcionalidades sem referência ou exemplificação de algum jogo existente, como Apostolos Ampatzoglou e Alexander Chatzigeorgiou (2006).

Atento a isso, surge um questionamento natural sobre a eficácia dos padrões GoF para solucionar funcionalidades de jogos digitais modernos. A resposta à essa dúvida pode direcionar cursos de desenvolvimento de jogos digitais sobre a relevância do ensino dos padrões GoF e empresas também do segmento de jogos digitais, sobre o investimento dado a aprendizagem dos padrões GoF aos seus desenvolvedores.

1.2.1 Objetivo geral

Analisar, através de um estudo da literatura, a pertinência do uso dos padrões de projetos GoF na solução de funcionalidades de jogos digitais modernos.

1.2.2 Objetivos específicos

Para alcançar o objetivo geral desse trabalho, é preciso alcançar os seguintes objetivos específicos:

- Descrever a estratégia, os critérios e o processo da realização da revisão sistemática da literatura;
- Analisar trabalhos que solucionem funcionalidades de jogos digitais através do uso de padrões de projeto GoF;
- Para cada padrão de projeto, procurar atributos e métricas de qualidade de software;
- Para cada padrão de projeto GoF que for encontrada uma aplicação para solucionar uma funcionalidade de jogo digital, procurar uma equivalência desta funcionalidade com a de um jogo digital moderno e descrever essa relação.

1.3 ESTRUTURA DA DISSERTAÇÃO

O trabalho trata de uma revisão sistemática que envolve os contextos de jogos digitais e padrões de projeto. Assim, a estrutura é dada da seguinte forma:

- **Capítulo 2: Fundamentação Teórica**: descreve em dois subtópicos o que é e como se dá o protocolo de uma revisão sistemática, e, quais sãos os atributos e as métricas de qualidade de software usadas para os padrões de projetos GoF;
- **Capítulo 3: Metodologia de Pesquisa**: explica como foram criadas as perguntas as quais este trabalho busca sanar, bem como apresenta qual foi a estratégia adotada para a escolha da string de busca usada nos motores de busca, quais foram os critérios de seleção dos trabalhos obtidos nos resultados dos motores de busca, e, por fim, as informações extraídas após a análise dos trabalhos;
- **Capítulo 4: Padrões de Projeto GoF**: São descritos cada um dos 23 padrões de projetos GoF agrupados em 3 categorias, cada uma delas contendo a tabela que aponta os trabalhos incluídos que abordam os padrões da categoria em questão. Também, para cada padrão há uma estrutura de subtópicos:

    - **Descrição do padrão de projeto GoF**: descreve a definição do padrão de projetos, bem como sua estrutura base, sua aplicabilidade e quais atributos de qualidade e métricas de software são atribuídas a ele.
    - **Uso em Jogos Digitais**: descreve uma implementação do padrão de projetos citado, encontrada na literatura, onde ele é utilizado para solucionar uma funcionalidade de jogo digital.
    - **Equivalência com Jogo Digital Moderno**: apresenta uma funcionalidade de um jogo digital moderno equivalente, em termos de abstração, com a funcionalidade citada no subtópicos anterior, afim de avaliar a aplicabilidade deste padrão de projetos com a funcionalidade do jogo moderno.
- **Capítulo 5: Conclusão**: Por fim, dá-se as últimas discussões e considerações sobre o trabalho, como limitações da pesquisa e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são dadas as fundamentações teóricas sobre revisão sistemática da literatura e atributos e métricas de qualidade de software.

2.1 REVISÃO SISTEMÁTICA DA LITERATURA

A revisão sistemática da literatura é um método para pesquisas que surgiu na medicina, mas foi adotado por diversas áreas de conhecimento, como a Engenharia de Software. Seu protocolo trata de identificar, interpretar e avaliar um determinado domínio, seja ele um tema, fenômeno ou pergunta (KITCHENHAM et al., 2007). Travassos e Biolchini (2007) reforçam que a revisão sistemática da literatura é uma forma abrangente e não tendenciosa para atender pesquisas de cunho científico.

Para Domholdt (2005) a revisão sistemática da literatura deve possuir três itens essenciais: definir o objetivo da pesquisa de forma clara; recolher a literatura; e, avaliar os estudos que possivelmente atendem aos objetivos da pesquisa. De modo similar, Travassos e Biolchini (2007) elegem três itens essenciais:

- Planejamento: são inclusos os principais objetivos da pesquisa, as questões específicas a serem respondidas, os critérios de inclusão ou exclusão de trabalhos, a estratégia de pesquisa adotada, a primeira escolha de trabalhos inclusos, a seleção final dos trabalhos a serem analisados e, por fim, a abordagem da revisão.
- Execução: nessa fase ocorre a investigação nas fontes em busca dos trabalhos que atendem ao objetivo da pesquisa, para que sejam analisados e classificados como inclusos ou exclusos. É possível que hajam adaptações do protocolo afim de transpor limitações identificadas nesse processo.
- Análise de Resultados: os dados dos trabalhos inclusos são extraídos para então serem analisados e sintetizados conforme o protocolo.

Para Castro (et al., 2002) a Análise de Dados inclui não apenas a descrição do trabalho e seus resultados, mas também a qualidade do mesmo. Eles propõem que a análise seja dividida em Análise Estatística e Análise de Qualidade. Na estatística é possível várias meta-análises, enquanto a de qualidade intenciona a detecção de semelhanças e diferenças entre os trabalhos. Uma vez concluída essa

fase, é feita a Interpretação dos Dados, fase essa de alta complexidade, uma vez que mesmo com o protocolo a ser seguido, a subjetividade é inevitável.

2.2 JOGOS DIGITAIS

Apesar da aparente facilidade que as pessoas têm para identificar o que é um jogo, há na literatura grande dificuldade para definir o que é um jogo. Entre diversas propriedades levantadas entre vários autores, há concordância entre algumas delas. O autor Ewerton Menezes (2012) avalia essa relação entre as definições de Huizinga (2000), Schell (2010) e Schreiber (2009).

Tabela 1 – Relação entre propriedades dos jogos

	Huizinga	**Schell**	**Schereiber**	**Análise**
Atividade livre	Cita	Cita	Cita	Mesmo uma atividade imposta pode ser vista de maneira lúdica e consequente mente se transformar em um jogo.
Sistema fechado e formal	Cita.	Cita.	Cita.	Os limites do jogo através de um sistema fechado e formal não impede que abranja uma parte do cotidiano.
Regras	Cita.	Cita.	Cita.	As regras definem o sistema. Para Huizinga produzem ordem, para Schereiber ineficiência.
Fuga	Exterior a vida cotidiana.	Valores internos.	Artificial e seguro.	O sistema fechado provê uma separação confortável do mundo real. Entendemos como uma consequência de um sistema fechado.
Envolvente	Cita	Cita.	-	O sentimento de envolvimento do jogador é uma consequência.
Valor material	Cita.	-	Cita.	Os autores citam como desprovido de valor material. No entanto, em MMORPG os bens materiais são muito valorizados.
Objetivo	-	Cita.	Cita.	O sentido de objetivo está relacionado com uma definição de estado final do jogo. No RPG não temos, necessariamente um final, nem a definição de um final.
Conflitos	-	Cita.	Cita.	O conflito é a força contrária.

Interatividade	-	Cita.	Cita.	Interatividade define melhor do que decisão. Alguns jogos possuem interatividade mas não escolhas.
Incerteza	-	Cita.	Cita.	A incerteza é uma característica dos jogos.
Arte	-	-	Cita.	A expressão artística está em elementos como música ou narração áudio visual.
Não sério	Cita.	-	-	Jogos podem ser utilizados para outras atividades que não sejam entretenimento. Por exemplo, simuladores para treinamento.
Derrota e Vitória	-	Cita.	-	Nem todos os jogos possuem uma métrica de vitória definida ao final do jogo. Por exemplo, RPG.

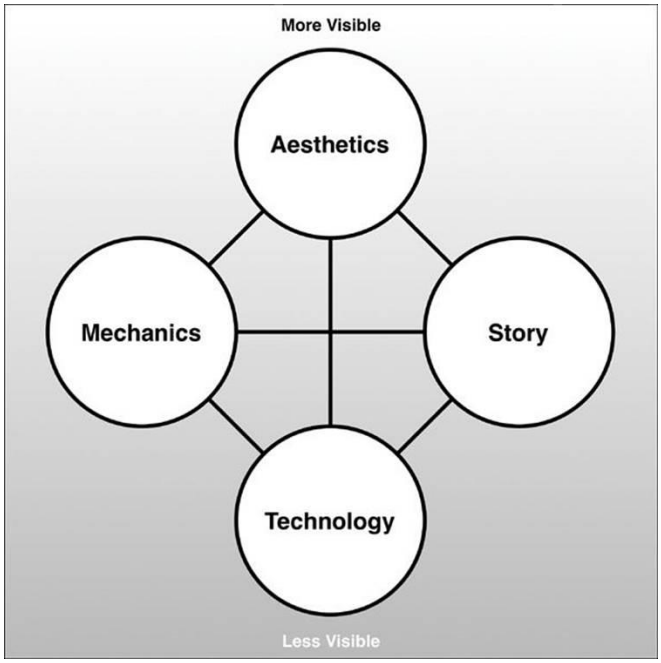
Fonte: Ewerton Menezes (2012). Análise acerca das semelhanças entre as propriedades dos jogos definidas por Huizinga (2000), Schell (2010) e Schreiber (2009).

Entendendo a base conceitual dos jogos, é necessário também entender o que é um software para concluirmos o que é um jogo digital. Sommerville (2011) define softwares como sendo “programas de computador e documentação associada. Produtos de software podem ser desenvolvidos para um cliente específico ou para o mercado em geral”. O autor também explica que há vários tipos de softwares e cada qual possui uma abordagem diferente para seu desenvolvimento. Abordagens essas totalmente diferentes entre um sistema controlador para instrumento científico e um sistema de informações corporativo. Ele conclui que ambas abordagens também não têm muito em comum com a abordagem para um jogo digital. Todos esses softwares precisam atender a requisitos da engenharia de software, porém, as técnicas usadas não devem ser necessariamente as mesmas.

Para Ampatzoglou e Stamelos (2010) o desenvolvimento de jogos digitais é substancialmente diferente do desenvolvimento clássico de softwares por terem um ciclo de desenvolvimento muito rápido para atenderem à janela de oportunidade do mercado. Por essa razão, a procura por metodologias para desenvolvimento de software voltadas para as especificidades dos jogos digitais tem crescido bastante.

Além da abordagem de desenvolvimento, os jogos se diferenciam de sistemas não-jogos por terem como um dos objetivos fundamentais o entretenimento do usuário. Assim, podemos entender os jogos digitais como sendo uma categoria dos sistemas de softwares. O jogo digital então se embasa nos conceitos de jogos, fazendo uso dos elementos digitais para atender a seus requisitos. Shell (2010) define quatro elementos básicos dos jogos, demonstrados na figura 3. Na perspectiva de Schell, os jogos digitais são jogos onde o elemento *technology* é um computador – seja ele um computador pessoal, smartphone, console etc.

Figura 2-1 – Quatro elementos dos jogos



Fonte: Shell (2010). O elemento *technology* especifica em qual plataforma o jogo é disposto, como tabuleiros ou consoles.

2.3 ATRIBUTOS E MÉTRICAS DE QUALIDADE DE SOFTWARE

Para avaliação de atributos e métricas de qualidade de software, os autores Ampatzoglou, Charalampidou e Stamelos (2013) levantam 14 propriedades dos padrões de projeto GoF, onde 8 são atributos de qualidade de software e 6 são métricas de software. Os autores mapeiam trabalhos que relacionam de forma positiva ou negativa a propriedade em cada padrão de projeto.

Os 8 atributos de qualidade de software analisados nos padrões de projeto GoF por Ampatzoglou, Charalampidou e Stamelos são:

- **Adaptabilidade**: é a facilidade em ser adaptado para outro ambiente sem grades mudanças (ISO/IEC 25010: 2011);
- **Compreensibilidade**: Grau do quanto um código pode ser compreendido facilmente (KHOMH e GUÉHÉNEUC, 2008);
- **Confiabilidade**: o quão o desempenho se mantém no nível estável sob as condições estabelecidas (ISO/IEC 25010: 2011);
- **Estabilidade**: Grau do quanto um sistema pode lidar com uma grande quantidade de dados e cálculos em tempo de execução (KHOMH e GUÉHÉNEUC, 2008);
- **Manutenibilidade**: O quão é possível evoluir para refletir a mudança de requisitos ou melhorias identificadas (SOMMERVILLE, 2011).
- **Portabilidade**: enquanto adaptabilidade trata do grau da mudança, a portabilidade trata da quantidade de ambientes diferentes em que pode ser transferido sem a necessidade de adaptação (ISO/IEC 25010: 2011);
- **Reusabilidade**: O grau do quanto uma parte de um software pode ser reutilizada em outro software (KHOMH e GUÉHÉNEUC, 2008);
- **Usabilidade**: Trata do quão é atrativo e fácil de utilizar o código (AMPATZOGLOU, CHARALAMPIDOU e STAMELOS, 2013).

As 5 métricas de qualidade de software analisadas nos padrões de projeto GoF por Ampatzoglou, Charalampidou e Stamelos (2013), bem como suas definições, são:

- **Acoplamento**: são utilizados três filtros:
    - Fator de acoplamento: número de acoplamentos sem herança;
    - Acoplamento eferente: a quantidade de tipos dentro de um pacote que dependem de outros tipos que estão fora deste package.
    - Acoplamento entre objetos: representa o número de classes a qual uma classe está acoplada, como uso de variáveis, métodos, tipos de retorno e throws. Tipos primitivos são desconsiderados;
- **Coesão**: Foram utilizados dois filtros:
    - Relacional: o número médio de relacionamentos internos para cada tipo de pacote.
    - Ausência de coesão em métodos: indica o quão os métodos estão acoplados a outros, resultando em baixa reusabilidade;

- **Complexidade**: são utilizados três filtros:
    - Quanto maior a quantidade de métodos em uma classe, mais complexa ela é;
    - Complexidade ciclomática, que é a soma de todas as estruturas de repetição, como “for” e “while”;
    - A soma de cada atributo da classe, considerando o peso diferente para alguns, como arrays;
- **Herança**: são utilizados quatro filtros:
    - Quão profunda a classe está na árvore de herança;
    - A quantidade de classes filhas desta classe. A inexistência de filhas facilita a reusabilidade dessa classe;
    - A proporção entre classes e interfaces abstratas e o número total de tipos (classes concretas e interfaces concretas) do package.
    - Porcentagem de operações não-redefinidas em relação ao número de operações herdadas;
- **Polimorfismo**: quantidade de métodos que utilizam polimorfismo.
- **Tamanho**: são utilizados dois filtros:
    - Quantidade de linhas de código;
    - Quantidade de classes.

Uma vez que cada padrão de projeto GoF pode possuir uma quantidade de trabalhos citando uma mesma propriedade positivamente e negativamente, para todos os padrões de projetos GoF descritos nesse trabalho serão apontados apenas as propriedades que aparecem com efeito positivo uma quantidade de vezes maior do que os trabalhos que apresentam a mesma propriedade com efeito negativo. Assim, os que apresentam a mesma quantidade de trabalhos apontando uma propriedade tanto com efeito positivo quanto negativo, também não serão citadas no dado padrão de projetos.

3 METODOLOGIA DE PESQUISA

Essa capitulo descreve todo o procedimento e ciclo da revisão sistemática da literatura utilizada nesse trabalho.

3.1 PROTOCOLO DE PESQUISA

O protocolo o qual essa pesquisa segue é dado da seguinte forma:

- São definidas as perguntas das quais a pesquisa objetiva responder;
- É definida a estratégia de busca a ser usada, as fontes utilizadas, os termos principais ao tema e as strings aplicadas nas fontes;
- Os resultados são colhidos e é feita uma extração dos dados desses resultados. Por fim, é feita uma síntese dos dados extraídos.

3.2 PERGUNTA DE PESQUISA

Uma vez definidos os objetivos da pesquisa, foi levantada a pergunta principal desse trabalho: Os padrões de projetos GoF são relevantes no desenvolvimento de jogos digitais modernos?

3.2.1 Pergunta específicas de pesquisa

Para alcançar a resposta da pergunta principal, é necessário responder outras perguntas de maior granularidade:

- Quais padrões de projeto GoF são utilizados em propostas de soluções para funcionalidades de jogos digitais?
    - Há equivalência entre as funções solucionadas na literatura e funções de jogos digitais modernos?
- Quais padrões de projetos GoF possuem avaliações de atributos de qualidade e métricas de software quando aplicados em jogos digitais?

Uma vez que moderno é um termo qualitativo que necessita de uma definição numérica, foi, pela experiência do pesquisador como usuário e desenvolvedor de jogos digitais, considerado como “jogo moderno” qualquer jogo digital publicado a partir de 2017 na plataforma Steam, que já apresenta uma grande quantidade de

jogos, suficiente para abordar o objetivo da pesquisa. A plataforma de divulgação e comercialização de jogos digitais – Steam, foi escolhida para fins de delimitação do alcance de busca desses jogos, considerando que ela contempla a presença de aproximadamente dez milhões de usuários simultâneos diariamente e dispõe de milhares de jogos disponíveis para download (MOGNON, 2018).

3.3 ESTRATEGIA DE BUSCA

Para realizar a pesquisa de trabalhos nas bases de dados, foi utilizada uma string de busca procurando por palavras-chave relacionadas ao tema. A string foi inserida em todas os motores de busca das fontes selecionadas, cada qual com suas particularidades. Também, adicionalmente foram feitas buscas manuais tanto sobre o uso de padrões de projeto GoF em jogos digitais, quanto atributos de qualidade e métricas de software.

3.4 CRITÉRIOS DE INCLUSÃO E EXCLUSÃO

Todos os trabalhos devem atender a uma lista de critérios para serem inclusos ou exclusos. A análise é feita a partir do título, das palavras chaves e resumo de cada trabalho, atendendo aos seguintes itens:

- Tratem sobre:
    - Atributos de qualidade de software em padrões de projetos GoF, ou
    - Métricas de software em padrões de projetos GoF, ou
    - Propostas de solução de funcionalidade de jogo digital utilizando um padrão de projetos GoF.
- Estejam escritos na língua portuguesa ou inglesa;
- Publicados no período de Janeiro de 1980 até Abril de 2018.

3.5 FONTES DE BUSCA

A seleção dos motores de busca foi feita manualmente, todas suportando a busca por palavras-chave.

- ACM

- Biblioteca Digital Brasileira de Teses e Dissertações
- IEEE Xplore
- Science Direct

3.6 TERMOS DE BUSCA

Baseado nas questões de pesquisa foi identificado o tema em português “jogos com padrões de projeto”, e, em inglês, “games with design patterns”. Assim, para a busca por trabalhos através das palavras-chave, foram atribuídos os seguintes termos:

Para o contexto de jogos digitais, em português:

- Jogo
- Videogame
- Video-game
- Video game

Para o contexto de jogos digitais, em inglês:

- Game
- Videogame
- Video-game
- Video game

Para o contexto de Padrões de Projeto GoF, temos em português o termo “Padrão de Projetos”, e, em inglês, “Design Pattern”.

3.7 STRING DE BUSCA

Com os termos definidos e os motores de busca selecionados, duas strings de busca foram construídas. Uma para os motores de busca em português e outra para os motores de busca em inglês. Atento às especificidades da sintaxe de cada motor, foram criadas as seguintes strings:

- Biblioteca Digital Brasileira de Teses e Dissertações (português):

("jogo" OR "game" OR "videogame" OR "video-game" OR "video game" OR "jogos" OR "games" OR "videogames" OR "video-games" OR "video games") AND ("padrão de projetos" OR "padrões de projetos" OR "padrão de projeto")

- ACM (inglês):

keywords.author.keyword:(+("design pattern" "design patterns") +("game" "videogame" "vídeo game" "video-game"))

- IEEE Xplore (inglês):

(("Author Keywords":"design pattern" OR "Author Keywords":"design patterns") AND ("Author Keywords":"game" OR "Author Keywords":"games" OR "Author Keywords":"videogame" OR "Author Keywords":"video game" OR "Author Keywords":"video-game"))

- Science Direct (inglês):

("design pattern" OR "design patterns") AND ("game" OR "games" OR "videogame" OR "video game" OR "video-game")

3.8 RESULTADOS

Realizando a busca por strings, a primeira e única aplicação da string em português foi o motor de busca da Biblioteca Digital Brasileira de Teses e Dissertações, que trouxe 3 resultados. Em seguida, deu-se início ao uso das strings em inglês. O motor de busca da Science Direct trouxe 14 resultados, e, a IEEE Xplore trouxe 13 resultados. A ACM revelou 63 resultados, onde 3 deles não puderam ser obtidos (apêndice B). Dos 93 trabalhos, 4 estavam duplicados. Assim, 86 foram para o processo de seleção.

3.9 SELEÇÃO DOS RESULTADOS

Aplicados os critérios de inclusão e exclusão ao término da pesquisa, foram selecionados 11 dos 86 trabalhos obtidos da pesquisa automática por strings. Para ampliar a quantidade de trabalhos inclusos, a pesquisa manual trouxe 9 trabalhos também atendendo aos critérios. Assim, houve um total de 20 trabalhos inclusos.

Tabela 2 – Quantitativo de trabalhos inclusos e exclusos

Descrição	**Quantidade**
Busca automática: trabalhos inclusos	11

Busca automática: trabalhos exclusos	75
Busca manual: trabalhos inclusos	9

Fonte: Jadson Almeida (2018).

3.10 EXTRAÇÃO DE DADOS

Para registro da extração, foi criada uma planilha com informações como título, autores, ano de publicação, meio de publicação e qual padrão de projeto GoF está implementado em um contexto de jogo digital. Também, uma segunda planilha foi criada para registro com título, autores, ano de publicação, meio de publicação, quais métricas de software ou atributos de qualidade de software são abordados e o padrão de projeto GoF que elas tratam.

3.11 SÍNTESE DE DADOS

Com a base da planilha criada na extração de dados, eles foram submetidos a uma análise afim de encontrar informações relevantes ao tema de pesquisa.

3.11.1 Síntese dos padrões de projetos gof

Entre os 20 trabalhos finais, 19 deles possuíam padrões de projetos GoF aplicados a jogos digitais. A análise identificou além do padrão utilizado, o contexto do jogo digital ao qual o padrão foi aplicado, para então identificar qual funcionalidade o padrão de projeto GoF de fato soluciona no trabalho em questão. Ao fim do processo, foram identificadas 119 propostas de implementações de padrões de projeto GoF para funcionalidades de jogos digitais. Essa síntese é demonstrada na Tabela 3.

Tabela 3 – Padrões de projetos GoF e Implementações em Jogos Digitais.

Padrão de Projeto GoF	**Quantidade de Implementações**
Chain of Responsibility	1
Interpreter	1

Adapter	2
Decorator	2
Façade	2
Iterator	2
Bridge	3
Builder	3
Mediator	3
Proxy	3
Flyweight	4
Memento	4
Template Method	4
Prototype	5
Abstract Factory	6
Composite	6
Visitor	6
Command	7
Singleton	7
Factory Method	9
Observer	10
State	14
Strategy	15

Fonte: Jadson Almeida (2018). Quantidade de funcionalidades de jogos digitais encontradas na extração de dados implementando cada padrão de projetos GoF.

3.11.2 Síntese dos atributos e métricas de qualidade de software

Com apenas dois trabalhos encontrados abordando atributos de qualidade de software e métricas de software, ambos sobre padrões de projetos GoF, em nenhum deles houve uma especificidade dada ao contexto de jogos digitais.

Um dos trabalhos é um mapeamento do impacto do uso de padrões de projetos GoF no desenvolvimento de softwares, sem definições se os softwares são jogos ou não-jogos. Esse mapeamento então, por apresentar uma grande quantidade de dados, é referenciado em todos os padrões de projetos GoF descritos

no próximo capítulo. O segundo trabalho que também aborda esse tema (PEKER & CAN, 2011), não terá a mesma atenção por não apresentar validações, argumentos ou referências sobre as afirmações feitas ao conferir atributos de qualidade de software e métricas de software à padrões de projetos GoF.

4 DESENVOLVIMENTO E RESULTADOS

Nesse capitulo são descritos os vinte e três padrões GoF, agrupados pelas categorias criacional, estrutural e comportamental. Com a totalidade dos dados de cada padrão, havendo uma funcionalidade implementada com um padrão de projetos GoF e uma funcionalidade equivalente em um jogo digital moderno, é possível concluir que este padrão permanece relevante para o desenvolvimento de jogos digitais modernos.

4.1 PADRÕES CRIACIONAIS

Os padrões criacionais referem-se à instanciação de objetos e classes. Eles abstraem o processo de instanciação e ajudam a deixar o sistema independente de como seus objetos são criados, compostos e representados. Um padrão criacional de classe utiliza herança para variar a classe que será instanciada. Um padrão criacional de objeto irá delegar a instanciação para outro objeto (GAMMA et al., 2007). Os trabalhos incluídos que utilizam padrões de projetos criacionais para solucionar funcionalidades de jogos digitais estão na tabela 4.

Tabela 4 – Trabalhos inclusos sobre padrões criacionais

Padrão	**IDs**
Abstract Factory	TI_01, TI_02, TI_09, TI_12, TI_13, TI_17
Builder	TI_12, TI_13
Factory Method	TL_03, TL_04, TL_07, TL_10, TL_11, TL_12, TL_13, TL_17, TL_19
Prototype	TL_01, TL_04, TL_08, TL_12, TL_13, TL_20
Singleton	TL_07, TL_08, TL_10, TL_12, TL_13, TL_15, TL_16

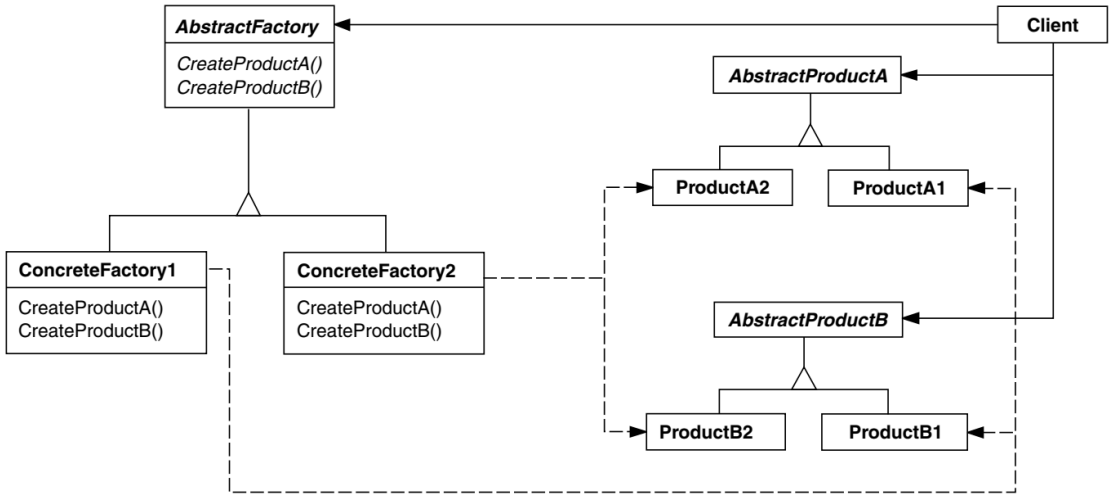
Fonte: Jadson Almeida (2018).

4.1.1 Abstract Factory

Concede uma interface que permite a criação de um grupo de objetos que estão relacionados ou que contêm dependência, sem que seja necessário especificar as classes concretas desses (GAMMA et al., 2007). O padrão de projetos

Abstract Factory assemelha-se ao padrão de projetos *Factory Method*, porém, com um maior nível de abstração que aumenta seu poder de escopo.

Diagrama 4-1 – Estrutura base do padrão de projetos *Abstract Factory*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013) cita trabalhos que apontam efeitos positivos do padrão de projetos *Abstract Factory* nas propriedades:

- Adaptabilidade
- Manutenibilidade
- Polimorfismo

O autor Erich Gamma (et al., 2007) atribui as seguintes aplicabilidades ao padrão de projetos *Abstract Factory*:

- É necessário que haja independência entre o sistema e como a geração de seus produtos;
- O sistema deve ser compatível com um produto que é composto de uma família de outros produtos;
- É necessário que restritivamente uma família de objetos seja usada em conjunto;
- É desejável criar uma biblioteca de classes de produtos sem revelar suas implementações.

4.1.1.1. Uso em Jogos Digitais

O padrão de projetos *Abstract Factory* é apresentado no contexto de jogos digitais pelos próprios autores dos padrões de projeto GoF (GAMMA et al., 2007). No exemplo é utilizado um jogo de labirinto, no qual as partes do labirinto, como paredes, salas e portas, são criados com o padrão de projetos *Abstract Factory*. Os autores utilizam duas fábricas de labirintos, uma para labirintos comuns e outra para labirintos encantados. Para os produtos, são criadas as seguintes abstrações e produtos concretos:

- Porta
    - Porta comum
    - Porta encantada
- Parede
    - Parede comum
    - Parede explodida
- Sala
    - Sala comum
    - Sala encantada
    - Sala com bomba

É possível criar uma série de combinações com as fábricas e seus respectivos produtos.

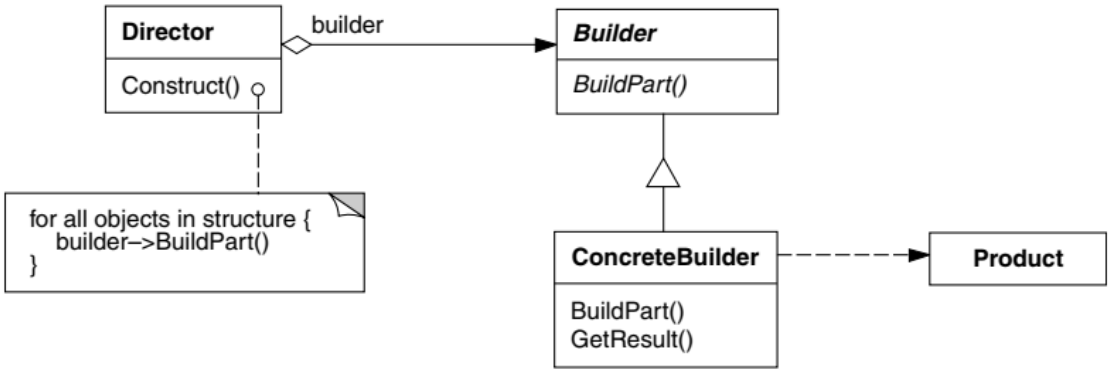
4.1.1.2. Equivalência com Jogo Digital Moderno

Em um construtor de cenários, a exemplo os que estão presentes nas plataformas de desenvolvimento de jogos *Unreal Engine* e *Unity*, é plenamente cabível o uso do padrão de projetos *Abstract Factory* de maneira similar à aplicação do mesmo padrão na construção de labirintos feita por Erich Gamma (et al., 2007). A estrutura é a mesma, porém com diferentes fábricas e abstrações, que atenderiam à demanda do jogo. A plataforma *Unreal Engine 4* foi utilizada para criação de jogos como *ARK*: *Survival* e *Foxhole*; e os jogos *Fall of Light* e *Unruly Heroes* foram desenvolvidos na plataforma *Unity*.

4.1.2 *Builder*

O padrão de projetos *Builder* intenciona construir diferentes representações através de um processo extraído de uma representação mais complexa (GAMMA et al., 2007). Através de uma classe chave, o padrão de projetos *Builder* permite criar objetos que utilizarão o todo ou parte dessa classe chave.

Diagrama 4-2 – Estrutura base do padrão de projetos *Builder*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Builder* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Compreensibilidade
- Manutenibilidade
- Polimorfismo

O padrão de projetos *Builder* possui as seguintes aplicabilidades (GAMMA et al., 2007):

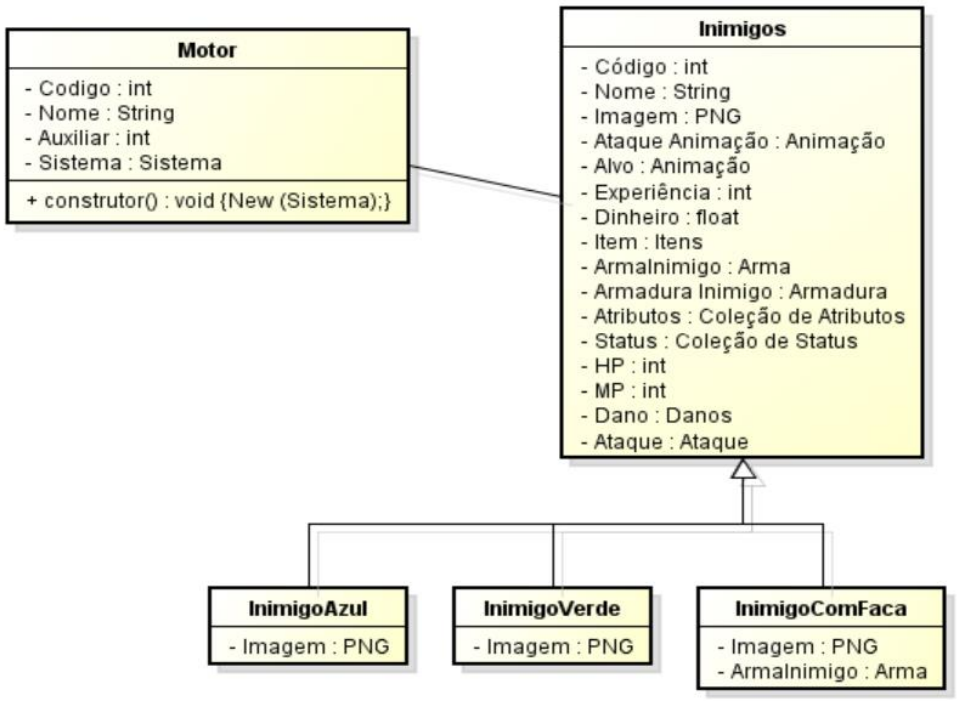
- O algoritmo de criação de um objeto precisa ser independente das etapas de construção e das partes que o compõem;
- É desejável que o processo de construção seja flexível para que o objeto construído provenha de uma variedade de representações.

4.1.1.3. Uso em Jogos Digitais

Segundo Perucia (et al., 2005) a criação de muitos inimigos diferentes resulta em baixa performance se tratando do uso de memória e disco primário, além de elevar o tempo de desenvolvimento do jogo.

Para solucionar a criação de uma grande variedade de inimigos, Roberto Figueiredo (2014) propõe o uso do padrão de projetos *Builder*. O autor utiliza como contexto o jogo "*Street of Rage*", onde o jogador andar por ruas lutando contra vários inimigos muito similares entre si. No Diagrama 4-3 Figueiredo demonstra como seria a funcionalidade sem o uso do padrão *Builder*.

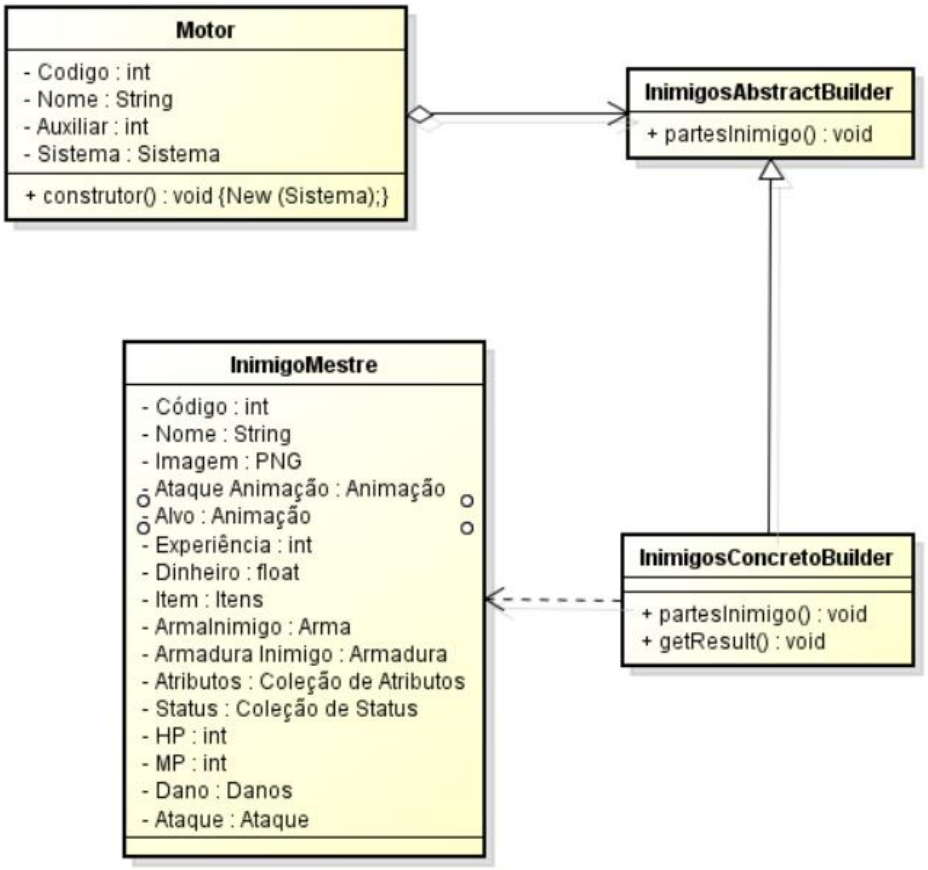
Diagrama 4-3 – Street of Rage sem o padrão de projetos *Builder*.



Fonte: Roberto Figueiredo (2014). Suposição da estrutura de inimigos do jogo *Street of Rage* sem o uso do padrão de projetos *Builder*.

Para a solução com o padrão *Builder*, Figueiredo propõe uma classe chave, denominada por ele como "mestre", que contém todos os métodos que dão os comportamentos possíveis aos inimigos. Dessa maneira, cada inimigo herda da classe "mestre" e utiliza, sobrescrevendo, os métodos que convém à cada inimigo, como demonstra o Diagrama 4-4.

Diagrama 4-4 – Street of Rage com o padrão de projetos *Builder*.



Fonte: Roberto Figueiredo (2014). Estrutura para criação dos inimigos do jogo *Street of Rage* utilizando o padrão de projetos Builder.

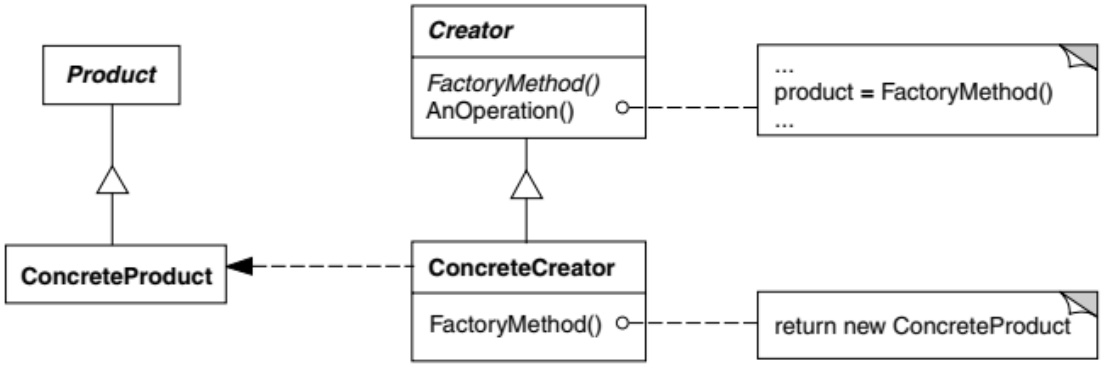
4.1.1.4. Equivalência com Jogo Digital Moderno

No jogo digital *Black Desert Online SA*, do gênero MMORPG, o jogador enfrenta diversos tipos de inimigos controlados pela inteligência artificial. Esses inimigos apresentam muitas similaridades de comportamento. Alguns, inclusive, parecem dividir o mesmo modelo 3D, mudando apenas a cor ou algum detalhe diferente – uma parte a mais ou a menos. Esse contexto é compartilhado com o exemplo de Figueiredo (2014). Os inimigos podem possuir um conjunto de comportamentos representados em uma classe “mestre” e os objetos concretos desses inimigos implementam as partes que os convém – o parcial do “mestre” ou mesmo sua totalidade.

4.1.3 *Factory Method*

O padrão *Factory Method* é do grupo de padrões criacionais. Sua intenção é “definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. Permite adiar a instanciação para subclasses” (GAMMA et al, 2007).

Diagrama 4-5 – Estrutura base do padrão de projetos *Factory Method*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Factory Method* nas propriedades:

- Adaptabilidade
- Manutenibilidade

O *Factory Method* possui as seguintes aplicabilidades (GAMMA et al., 2007):

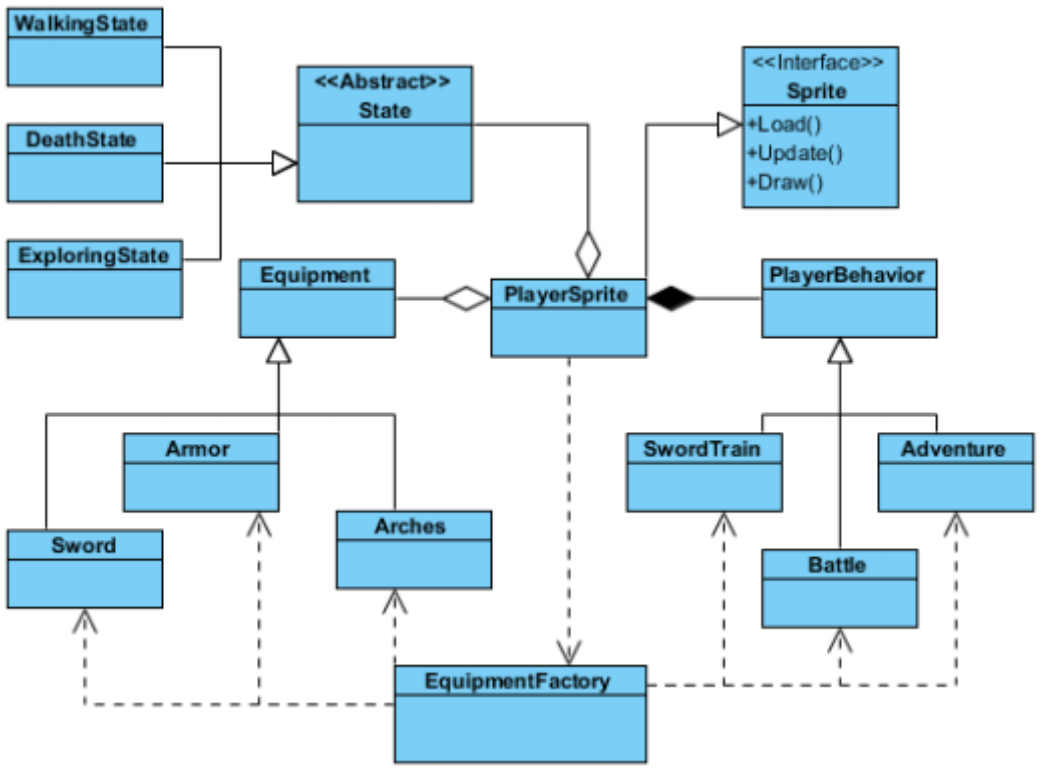
- Uma classe não pode antecipar a classe dos objetos que devem ser criados;
- Uma classe quer que suas subclasses especifiquem os objetos a serem criados;
- Classes delegam responsabilidade para uma dentre várias subclasses auxiliares, e você quer localizar o conhecimento de qual subclasse auxiliar tem a responsabilidade.

4.1.1.5. Uso em Jogos Digitais

O *Factory Method* é usado no artigo de Junfeng Qu, Yinglei Song e Yong Wei (2013) em um exemplo de jogo de RPG em que o jogador pode criar uma série de

equipamentos como espadas, arcos e armaduras. O uso do padrão é rapidamente descritivo e demonstrado em um diagrama que também engloba o uso, segundo o autor, dos padrões de projeto *State* e *Command*.

Diagrama 4-6 – RPG com os padrões de projetos *Command*, *Factory Method* e *State*.



Fonte: Junfeng Qu, Yinglei Song e Yong Wei (2013). Jogo de RPG em que é possível criar equipamentos através do padrão *Factory Method*. O diagrama também traz os padrões *State* e *Command*.

4.1.1.6. Equivalência com Jogo Digital Moderno

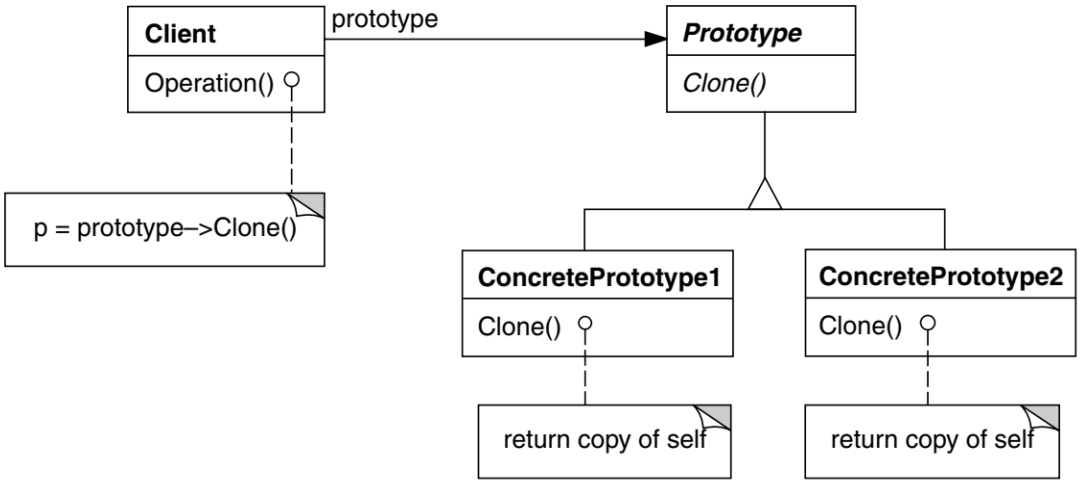
O jogo *Player's Unknows Battlegrounds* alcançou o título de jogo para PC mais vendido da história, ultrapassando a marca de quase 28 milhões cópias para PC, deixando para trás o jogo *Minecraft*. Até o momento dessa pesquisa, *Player's Unknows Battlegrounds* possui dois mapas distintos lançados. Há também uma lista de equipamentos para os personagens dos jogadores. Esses equipamentos são gerados no mapa de forma semi-aleatória. Porém, enquanto uma parte dos equipamentos está presente em ambos os mapas, outra parte está em apenas um dos mapas e não do outro. Ou seja, cada mapa tem uma lista própria de equipamentos possíveis.

Assim como Junfeng Qu, Yinglei Song e Yong Wei (2013) utilizam o *Factory Method* para criação de equipamentos para o personagem, também é possível aplicar o *Factory Method* na criação dos equipamentos para personagens no jogo *Player's Unknows Battlegrounds*. Assim, dada a geração aleatória de equipamentos, é preciso antes saber qual será o mapa da partida, atribuindo uma fábrica de equipamentos para cada mapa distinto. Nesse ponto identifica-se uma aplicabilidade do padrão *Factory Method*, que é uma classe não poder antecipar a classe dos objetos que devem ser criados.

4.1.4 *Prototype*

O padrão de projetos *Prototype* intenciona criar objetos através da cópia de instâncias de protótipos do objeto (GAMMA et al., 2007).

Diagrama 4-7 – Estrutura base do padrão de projetos *Prototype*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Prototype* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Adaptabilidade
- Compreensibilidade
- Manutenibilidade

Os autores Erich Gamma (et al., 2007) descrevem que o padrão de projetos *Prototype* possui as seguintes aplicabilidades:

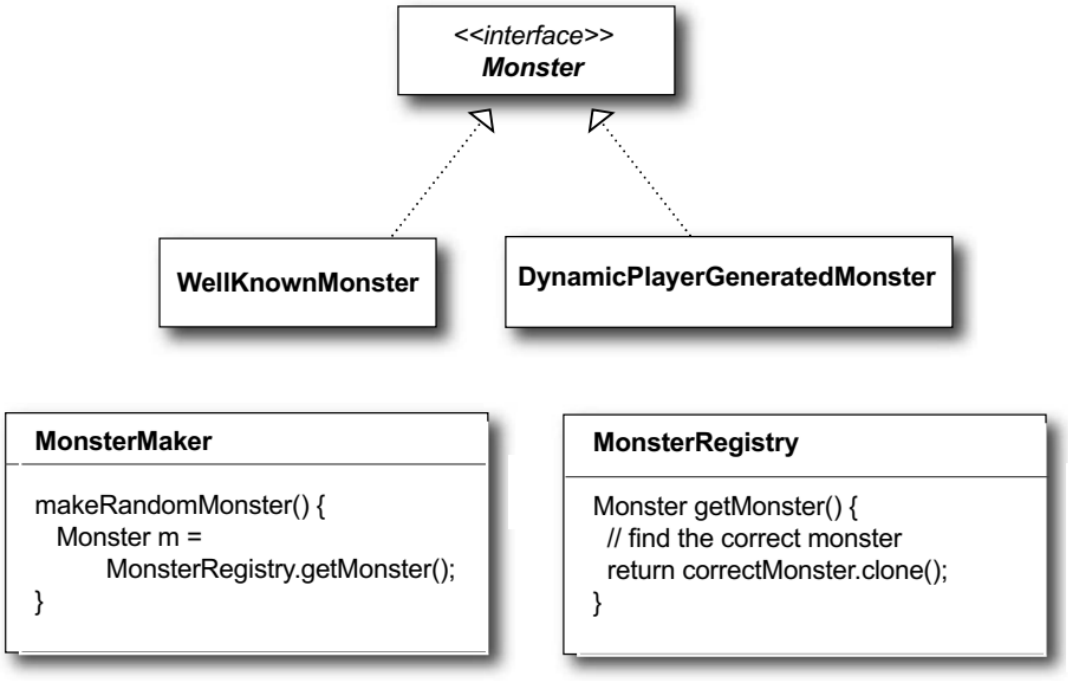
- É desejável que haja independente do sistema em relação à criação, composição e representação de seus produtos;
- As classes a serem instanciadas devem ser escolhidas em tempo de execução;
- Uma classe que será muito instanciada, possui poucas combinações diferentes de estado, sendo melhor dispor das possíveis instâncias e cloná-las quando necessário.

Segundo Freeman (et al., 2007), o padrão *Prototype* deve ser aplicado quando houver complexidade ou alto custo na criação de uma instância de uma classe.

4.1.1.7. Uso em Jogos Digitais

Freeman (et al., 2007) contextualiza um jogo em que uma dada aventura possui diversos inimigos monstros. Eles devem mudar conforme o cenário ao redor e os jogadores mais avançados podem criar suas próprias versões personalizadas de monstros.

Diagrama 4-8 – Criação de monstros com o padrão *Prototype*.



Fonte: Freeman (et al., 2007). Estrutura de criação de monstros que suporta monstros padrões (*WellKnownMonster*) e monstros personalizados pelo jogador (*DynamicPlayerGeneratedMonster*). A instância de monstro necessária é selecionada pelo algoritmo que clona essa instância.

No exemplo do Diagrama 4-8, o cliente não precisa saber da complexidade da criação dos monstros, nem o tipo do objeto a ser gerado. O cliente solicita um monstro e o algoritmo determina qual o tipo de instância deve ser clonado e o clone retornado para o cliente.

4.1.1.8. Equivalência com Jogo Digital Moderno

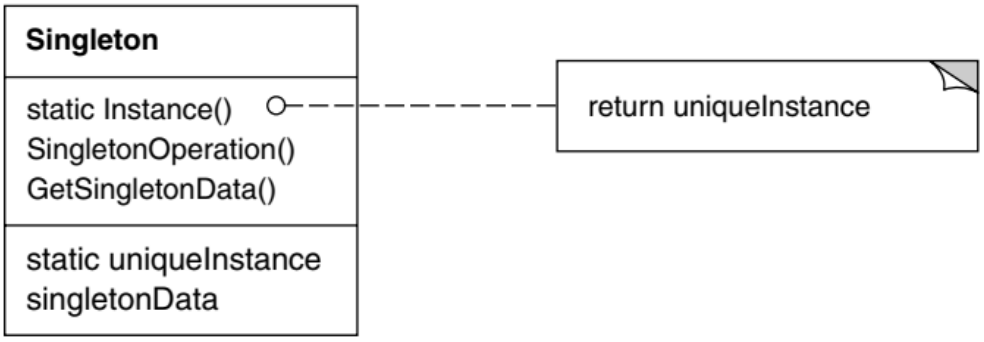
No jogo *Pro Evolution Soccer 2018*, é possível criar jogadores de futebol personalizados. É possível criar através de um jogador *default* ou de um jogador já existente em algum time ou seleção. Cada jogador de futebol tem uma grande lista de atributos, como: atributos numéricos (velocidade, resistência, qualidade do passe, força de chute etc.) e atributos compostos (drible especial que é acionado por um comando específico do jogador, “marcação homem-a-homem” que melhora as habilidades defensivas quando o jogador está marcando apenas um outro jogador, etc.); além disso, o jogador possui todo o design 3D, bem como animações diferenciadas para marcação de gol, cobrança de falta, corrida etc.

Com tantos atributos e comportamentos diferentes, a criação de um jogador de *Pro Evolution Soccer 2018* é deveras complexa. Nesse ponto, claramente podemos aplicar o padrão de projetos Prototype. Com ele podemos clonar a instância do jogador *default* ou a instância dos objetos responsáveis por comportamentos específicos, como: drible especial, marcação homem-a-homem, bicicleta, toque de letra etc. Devido a cada jogador possuir uma combinação própria desses comportamentos, é indicável que os comportamentos estejam instanciados e clonáveis para os jogadores que forem ser usados (atuar em uma partida, por exemplo).

4.1.5 *Singleton*

O padrão *Singleton* tem como atuação solucionar uma necessidade muito comum: conceder acesso global e único a um objeto (GAMMA et al., 2007).

Diagrama 4-9 – Estrutura base do padrão de projetos *Singleton*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Singleton* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Acoplamento
- Compreensibilidade

Segundo Erich Gamma (et al., 2007) o padrão de projetos *Singleton* possui as seguintes aplicabilidades:

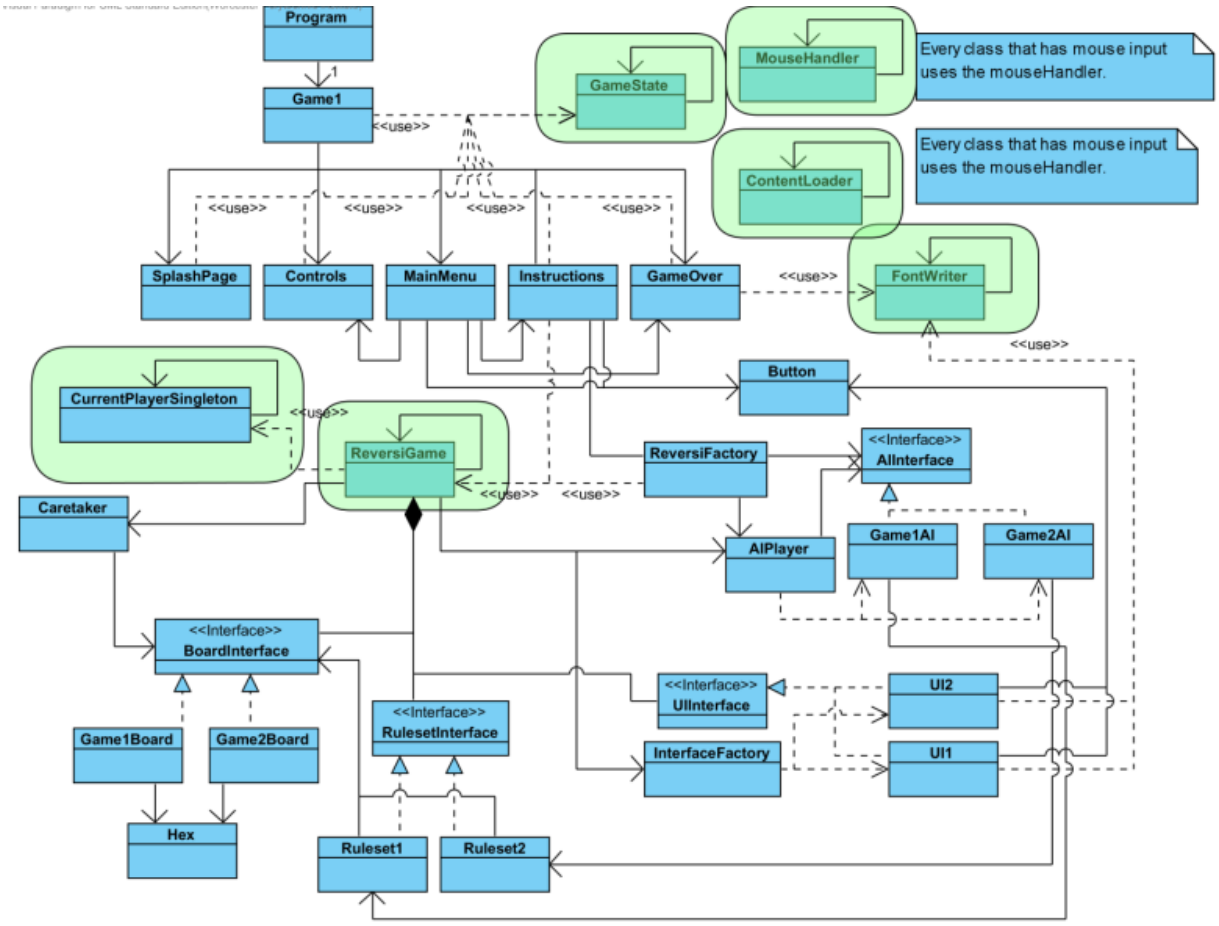
- É desejável que uma classe tenha apenas uma instância, e, que essa instância seja acessível por um ponto único e direto;
- O cliente precisa utilizar uma instância de uma classe estendida sem alterar o código da mesma.

4.1.1.9. Uso em Jogos Digitais

Utilizando diversos padrões de projeto GoF em seu trabalho com uma versão do jogo *Reversi*, como demonstra a estrutura no Diagrama 4-10, o autor Zawada (2013) aplica o *Singleton* em classes controladoras (GameState, *MouseHandler*, *ReversiGame* e *CurrentPlayerSingleton*) e classes auxiliares que proveem funcionalidades (*ContentLoader* e *FontWriter*). A classe *GameState* possui metodos de pintura de menu e atualização do estado do jogo; *CurrentPlayerSingleton* permite

saber qual é o jogador que está no turno ativo; *FontWriter* é usada para pintar letras na tela; *MouseHandler* captura os eventos do mouse e *ContentLoader* carrega na memória as imagens 2D necessárias.

Diagrama 4-10 – Versão do Reversi com o padrão de projetos *Singleton*.



Fonte: Zawada (2013). Estrutura de implementação de versão do jogo *Reversi*, utilizando diversos padrões de projetos GoF. As classes circuladas em verde são *Singletons*.

4.1.1.10. Equivalência com Jogo Digital Moderno

O padrão de projetos *Singleton* pode ser aplicado a qualquer jogo que possua um controlador, por exemplo. Pode ser um controlador de áudios com métodos para tocar um efeito ou trilha sonora; um controlador de chat entre jogadores ou um controlador de conexão com o servidor em um jogo online. O *Singleton* se mostra um padrão de grande utilidade, independentemente da idade da tecnologia utilizada.

4.2 PADRÕES ESTRUTURAIS

Os padrões estruturais intencionam interferir na forma com que classes e objetos são compostos para formar estruturas maiores (GAMMA et al, 2007). Eles facilitam o design da solução identificando uma maneira simples de realizar relacionamentos entre entidades. Os trabalhos que utilizam padrões de projetos estruturais para solucionar funcionalidades de jogos digitais são demonstrados na tabela 5.

Tabela 5 – Trabalhos inclusos sobre padrões estruturais

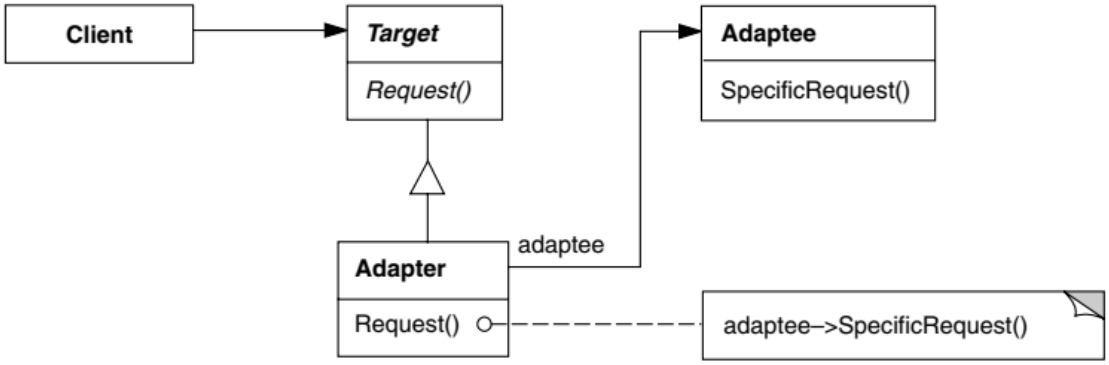
Padrão	**IDs**
Adapter	TL_05, TL_12
Bridge	TL_02, TL_04, TL_06, TL_12
Composite	TL_07, TL_09, TL_18
Decorator	TL_12
Façade	TL_12, TL_16
Flyweight	TL_07, TL_08, TL_12, TL_19
Proxy	TL_01, TL_12, TL_17

Fonte: Jadson Almeida (2018).

4.2.1 *Adapter*

Também conhecido como *Wrapper*, o padrão *Adapter* é um padrão estrutural que converte uma interface ou um objeto de uma classe em outra interface ou objeto no formato solicitado pelo cliente (GAMMA, et al., 2007).

Diagrama 4-11 – Estrutura base do padrão de projetos *Adapter*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Adapter* nas propriedades:

- Adaptabilidade
- Complexidade
- Manutenibilidade

O padrão de projetos *Adapter* possui as seguintes aplicabilidades (GAMMA et al., 2007):

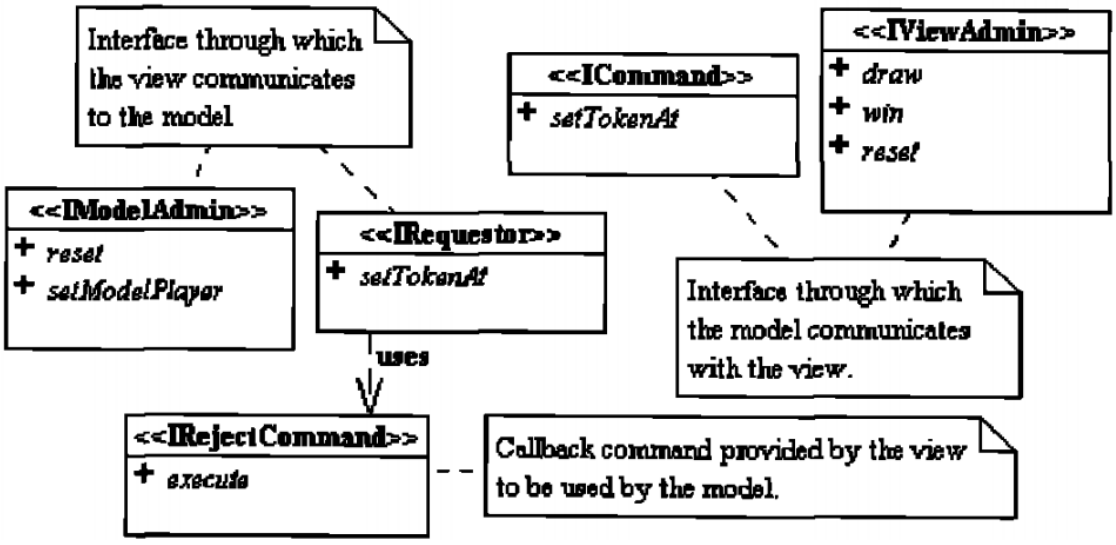
- É necessário mudar a interface, porém, mantendo a classe existente;
- Se deseja criar classes reutilizáveis que não possuem interfaces compatíveis.

4.2.1.1 Uso em Jogos Digitais

Semelhante à funcionalidade de desacoplamento entre o *model* do jogo e seu *view*, presentes no padrão *Façade* com os autores Gestwicki e Sun (2008), o *Adapter* é usado por Nguyen e Wong (2002) também na funcionalidade de comunicação entre *model* e *view*. Porém, aqui a arquitetura do jogo está em MVC (BUCANEK, J. 2009); e, na comunicação entre modelo e visão, o acoplamento é evitado através do controlador que intermedia a comunicação. Nesse caso os autores aplicam o padrão de projeto *Adapter* para ajustar as resoluções da visão. Os autores justificam o uso do padrão porque toda a classe é estendida, com exceção da funcionalidade adicional que trata justamente do ajuste da resolução. O Diagrama

4-12 não demonstra claramente o padrão de projeto *Adapter*, mas traz uma noção do contexto geral o qual ele é aplicado.

Diagrama 4-12 – *Adapter* na arquitetura MVC.

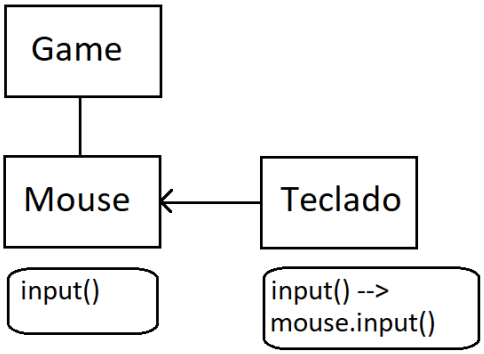


Fonte: Nguyen e Wong (2002). Contextualização da aplicação do padrão de projeto *Adapter* para ajustar a resolução da visão dentro da arquitetura MVC.

4.2.1.2 Equivalência com Jogo Digital Moderno

Independente do uso da arquitetura MVC na solução acima, o padrão de projeto Adapter pode ser aplicado de forma objetiva em um jogo que permite ao jogador utilizar diferentes tipos de controle como mouse, teclado ou joystick. Nesse exemplo, havendo uma implementação presente de um controle por mouse como a movimentação do cursor, o controle por teclado pode estender o controle do mouse e, recebendo os eventos de teclado, criar um evento *fake* para o objeto mouse, simulando o evento de movimentação do cursor através das teclas W, A, S, e D. O Diagrama 4-13 demonstra a estrutura exemplificada.

Diagrama 4-13 – *Adapter* convertendo *inputs*.

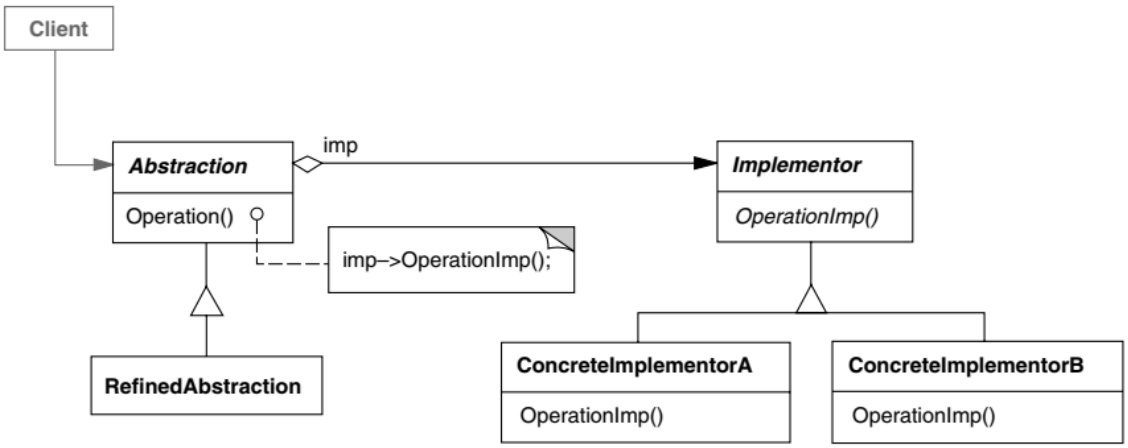


Fonte: Jadson Almeida (2018). Exemplificação de aplicação do padrão de projetos *Adapter* para simular a movimentação do cursor de mouse através do teclado.

4.2.2 Bridge

Também conhecido como "*handle/body*", o padrão de projetos *Bridge* tem como intenção desacoplar uma abstração de sua implementação, para que ambas possam variar de forma independente (GAMMA et al., 2007).

Diagrama 4-14 – Estrutura base do padrão de projetos *Bridge*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Bridge* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos em várias propriedades:

- Acoplamento
- Coesão
- Complexidade
- Compreensibilidade
- Estabilidade

- Herança
- Manutenibilidade

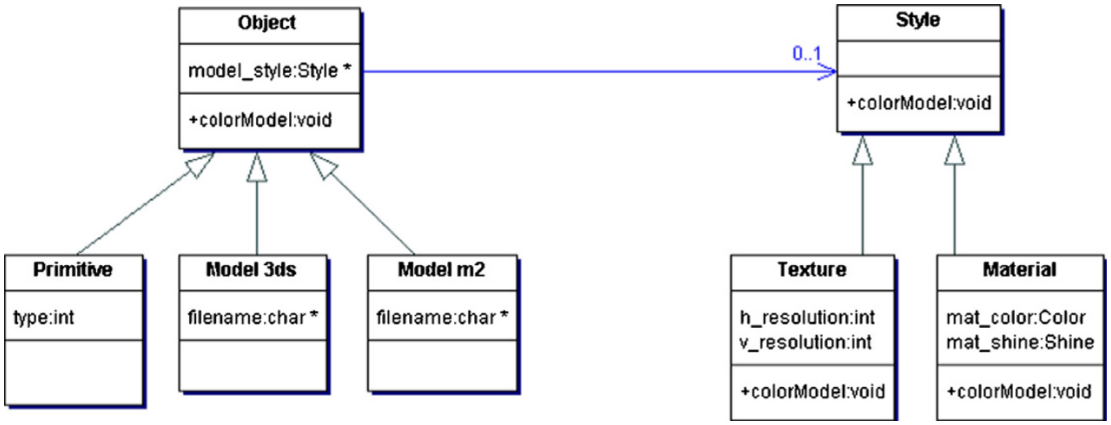
Segundo Erich Gamma (et al., 2007) o padrão de projetos *Bridge* tem as seguintes aplicabilidades:

- É desejável segregar permanentemente uma abstração de sua implementação;
- Estender abstrações e suas implementações de forma independente;
- As mudanças feitas na implementação de uma abstração não puderem, juntas com o restando do código a elas relacionado, ser recompiladas para o cliente;
- É desejável dividir a hierarquia de classes que devido à alta quantidade de extensões geradas por pequenas especificações;
- É desejável, de forma oculta ao cliente, compartilhar implementações entre múltiplos objetos.

4.2.2.1 Uso em Jogos Digitais

Os autores Ampatzoglou e Chatzigeorgiou (2006) propõem que o padrão *Bridge* pode ser usado por qualquer jogo com objetos 3D que podem ser coloridos de várias maneiras (texturização, aplicação de materiais etc.), pois os objetos 3D comumente variam entre tipos primitivos (esfera, cubo, círculo etc.) e modelos 3D exportados (arquivos .3ds, .m2, .fbx etc.). Como mostra o Diagrama 4-15, ao invés de um só objeto ser representar o modelo 3D e a coloração, o *Bridge* divide o objeto em dois, sendo "*Object*" e "*Style*", um para a lógica do modelo em si e outro respectivamente para a coloração deste modelo. Com a implementação de *Style* desacoplada da abstração *Object*, a estrutura é mais extensível, podendo ser criadas outras formas de coloração e de modelo 3D.

Diagrama 4-15 – *Bridge* e representações.



Fonte: Ampatzoglou e Chatzigeorgiou (2006). Estrutura da representação de objetos 3D implementada com o padrão de projetos *Bridge*, desacoplando o modelo 3D de sua coloração.

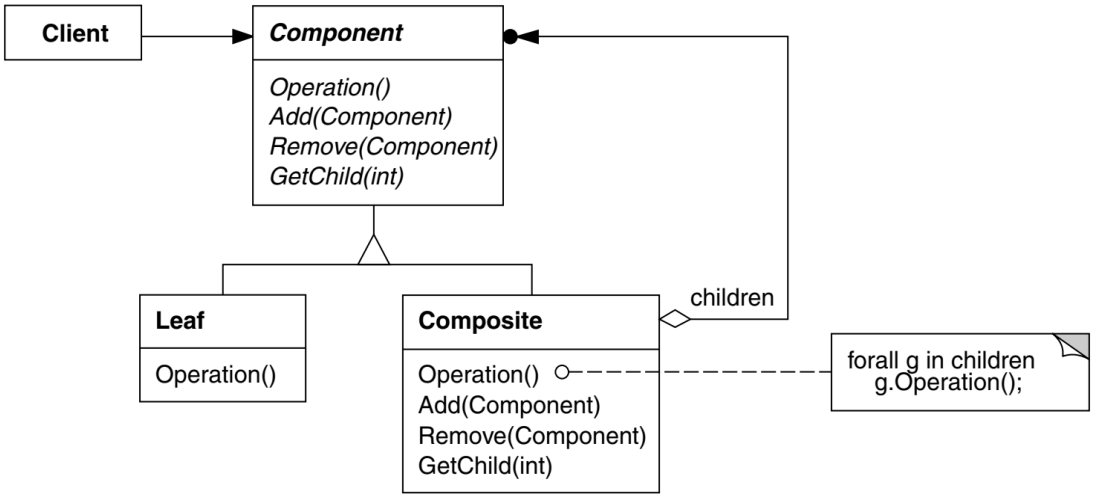
4.2.2.2 Equivalência com Jogo Digital Moderno

A aplicação do padrão *Bridge* pelos autores Ampatzoglou e Chatzigeorgiou (2006) é uma solução para um problema genérico presente em qualquer jogo 3D, independente de gênero ou de suas funcionalidades. A Steam apresenta os melhores lançamentos de 2017 com um total de 147 jogos, onde apenas 20 deles são em 2D e os demais 127 jogos são em 3D, indicando a alta relevância desta proposta para os jogos digitais modernos.

4.2.3 Composite

Utilizado para representar um objeto formado pela composição de objetos similares. Este conjunto de objetos pressupõe uma mesma hierarquia de classes a que ele pertence (GAMMA et al., 2007). Tal padrão é, normalmente, utilizado para representar em um objeto uma lista recorrente - ou recursiva - de objetos da mesma hierarquia do que contém a lista.

Diagrama 4-16 – Estrutura base do padrão de projetos *Composite*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Composite* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Adaptabilidade
- Manutenibilidade

Segundo Erich Gamma (et al., 2007) o padrão de projetos *Composite* possui as seguintes aplicabilidades:

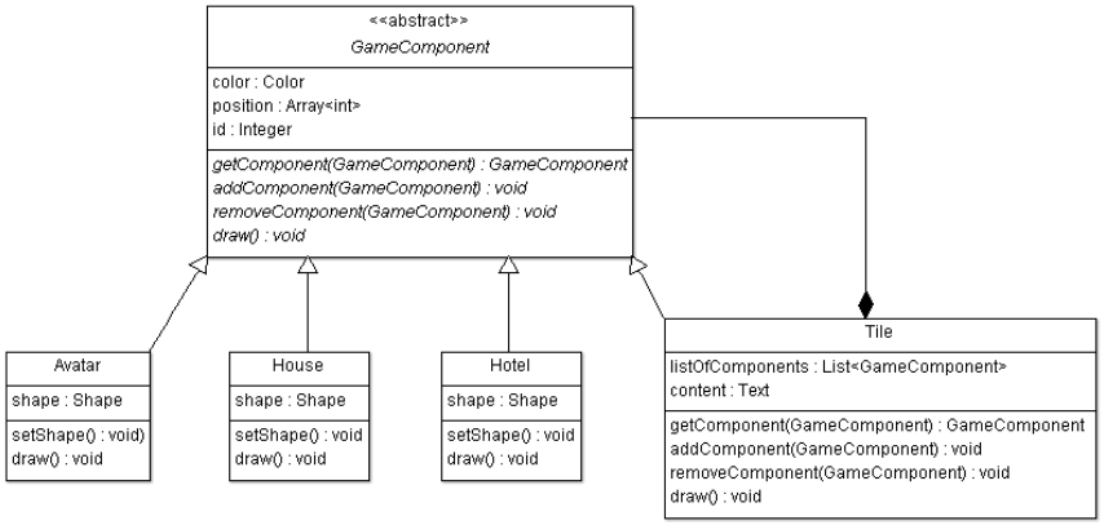
- É necessário criar uma estrutura hierárquica de árvore (raiz, folhas e galhos);
- É desejável que os clientes tratem uniformemente todos os objetos de uma mesma estrutura hierárquica.

4.2.3.1 Uso em Jogos Digitais

Utilizando o jogo *Monopoly*, com nome popular no Brasil de “Banco Imobiliário” (TORRES), é utilizado pelo autores Ampatzoglou, Kounoukla e Anagnostopoulos (2016) na implementação do padrão de projetos *Composite*. Como demonstrado no Diagrama 4-17, a classe abstrata *GameComponent* dá início à hierarquia que se estende às folhas (*Avatar*, *House* e *Hotel*) e ao composto (*Tile*). O *Tile* é cada campo que o *Avatar* se move na jogada de dados. O *Tile* pode estar vazio de composição ou conter vários *avatares* da partida e quantas *Houses* e

Hotels a regra do jogo permitir. Ou seja, há uma lista de folhas possíveis para cada *Tile*.

Diagrama 4-17 – *Composite* no jogo Banco Imobiliário.



Fonte: Ampatzoglou, Kounoukla e Anagnostopoulos (2016). Parte da estrutura do jogo *Monopoly* implementada com o padrão de projetos *Composite*. *Tile* é o objeto composto enquanto as folhas são *Avatar*, *House* e *Hotel*.

4.2.3.2 Equivalência com Jogo Digital Moderno

Em jogos como *SpellForce* 3, em que há a funcionalidade árvore de habilidades (*Skill Tree*), temos uma dependência entre as habilidades. Quando o jogador compra uma habilidade, ela pode habilitar a compra outras habilidades; essas outras habilidades, quando compradas, podem habilitar a compra de outra camada de habilidades; e assim sucessivamente, como demonstra a Figura 4-1.

Figura 4-1 – Árvore de habilidades do jogo *SpellForce* 3.

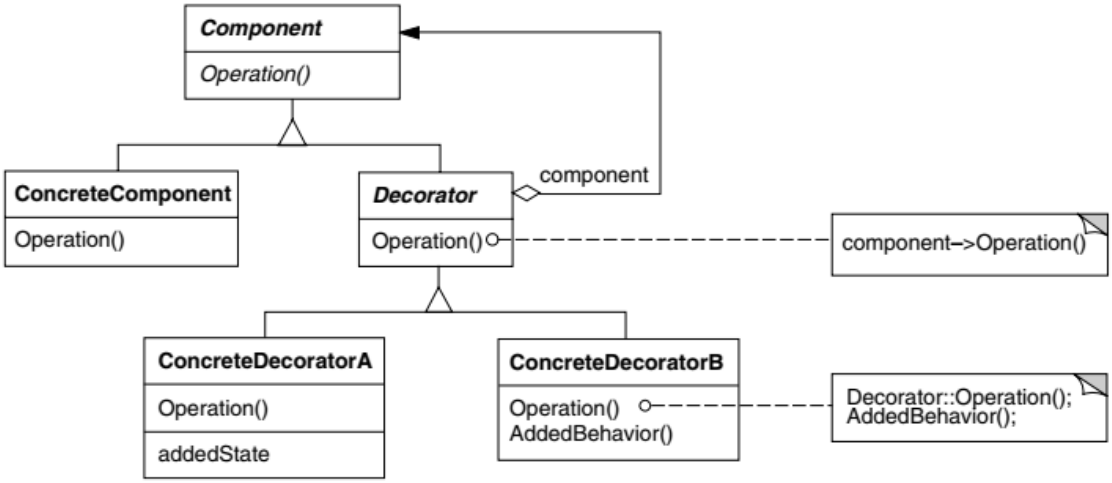
Fonte: *SpellForce* 3. Captura de tela do jogo, exibindo a tela de arvore de habilidades da progressão de um personagem. As dependências entre as habilidades são apresentadas de cima para baixo, ligadas por linhas.

As dependências entre as habilidades do personagem apontam ao padrão *Composite*. Na Figura 4-1, temos quatro arvores de habilidades: *Brutality*, *White Magic*, *Archery* e *Leadership*. No sentindo vertical, de cima para baixo, as últimas habilidades (1 no *Brutality* e *Leadership*, e, 2 no *White Magic* e *Archery*) são folhas, pois não são requisitos para nenhuma outra habilidade; todas as demais habilidades são compostas, pois possuem uma lista de possíveis habilidades que são dependentes dela – apenas comprando-a é possível habilitar a compra das dependentes.

4.2.4 Decorator

O padrão de projetos *Decorator* intenciona adicionar responsabilidades a um objeto dinamicamente, estender funcionalidades através de herança para agregar um conjunto delas nesse objeto (GAMMA et al., 2007). Através do Decorator é possível utilizar um método compartilhado por herança e combiná-lo em sequência com diferentes implementações do mesmo método.

Diagrama 4-18 – Estrutura base do padrão de projetos *Decorator*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Decorator* nas propriedades:

- Compreensibilidade
- Manutenibilidade

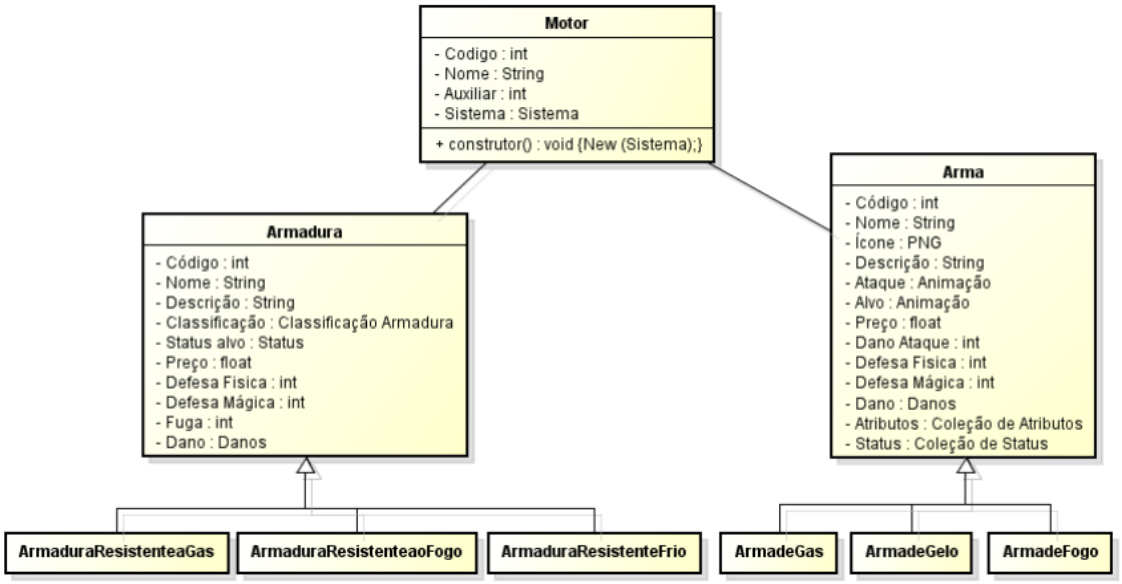
As aplicabilidades do padrão de projetos Decorator são, segundo Erich Gamma (et al., 2007):

- Acrescentar dinamicamente responsabilidades a um objeto sem afetar outros objetos;
- É desejável puder remover responsabilidades dinamicamente;
- Há uma grande quantidade de funcionalidade acrescidas e o uso de herança para atender a essa demanda é inviável.

4.2.4.1 Uso em Jogos Digitais

O autor Roberto Figueiredo (2014) argumenta que os jogos digitais do gênero RPG são um bom cenário para a implementação do padrão de projetos *Decorator*. Segundo o autor, o fato dos jogadores poderem customizar seus personagens de várias maneiras, em específico na atribuição de acréscimos de habilidades como a aquisição de novos itens que melhoram o ataque do personagem, é solucionável com o *Decorator*.

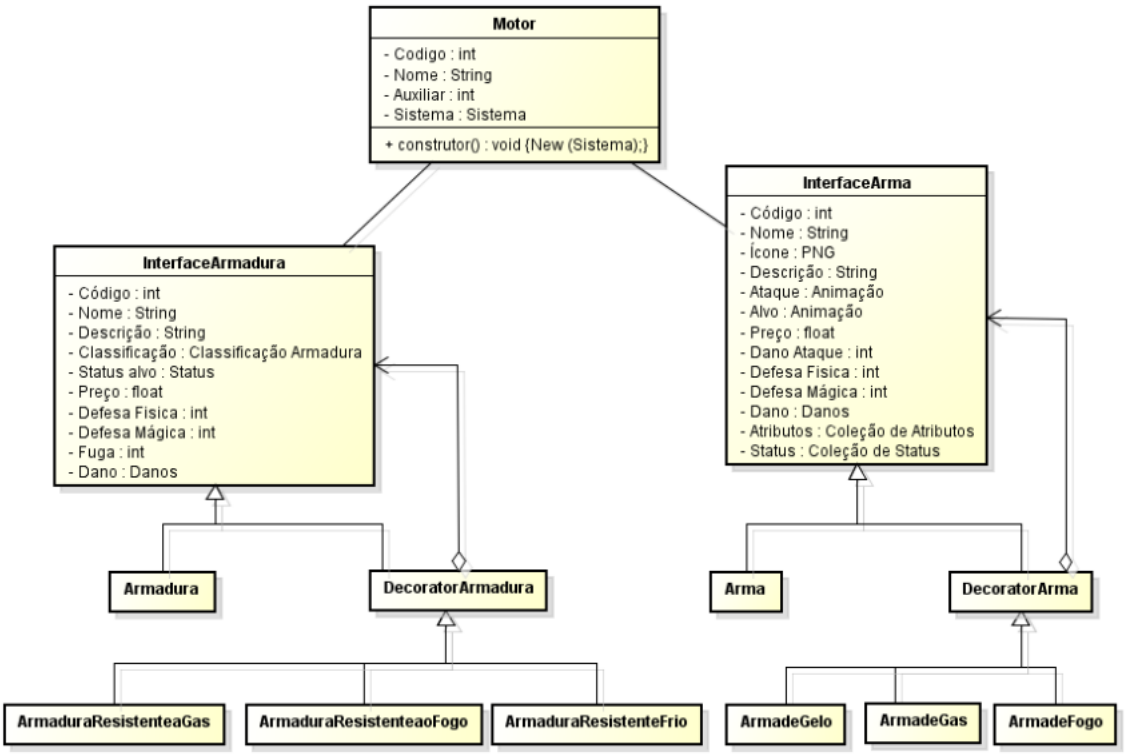
Diagrama 4-19 – Composição de equipamentos sem o padrão de projetos *Decorator*.



Fonte: Roberto Figueiredo (2014). Proposta de estrutura para a funcionalidade de composição de equipamento ou comportamento, criando uma variedade de combinações sem o uso de padrões de projetos.

No Diagrama 4-20, o autor demonstra como seria a estrutura dessa funcionalidade em um jogo do gênero RPG sem o uso do padrão de projetos *Decorator*.

Diagrama 4-20 – Composição de equipamentos com o padrão de projetos *Decorator*.



Fonte: Roberto Figueiredo (2014). Proposta de estrutura para a funcionalidade do Diagrama anterior, porém, utilizando o padrão de projetos *Decorator*. Tanto armadura quanto arma podem combinar uma quantidade ilimitada de atributos.

Ao invés de criar vários métodos, cada qual com uma combinação diferente de poderes, é possível criar apenas um método para cada poder, e então combiná-los na classe principal do comportamento. Assim, os atributos, tanto de arma quanto de armadura, podem ser combinados em quantas tipos o jogo exigir, como demonstra a estrutura no Diagrama 4-21.

4.2.4.2 Equivalência com Jogo Digital Moderno

No jogo digital *Player's Unknows Battlegrounds* os personagens podem utilizar armas que possuem acessórios encontrados em cada partida. Cada arma tem uma lista diferente de possíveis acessórios possíveis de serem anexados à arma. Cada acessório altera algum atributo ou comportamento da arma. Há, no momento desta pesquisa, 42 acessórios diferentes e 36 armas diferentes. Assim, a quantidade de combinações é imensa, pois algumas das armas podem conter até 6

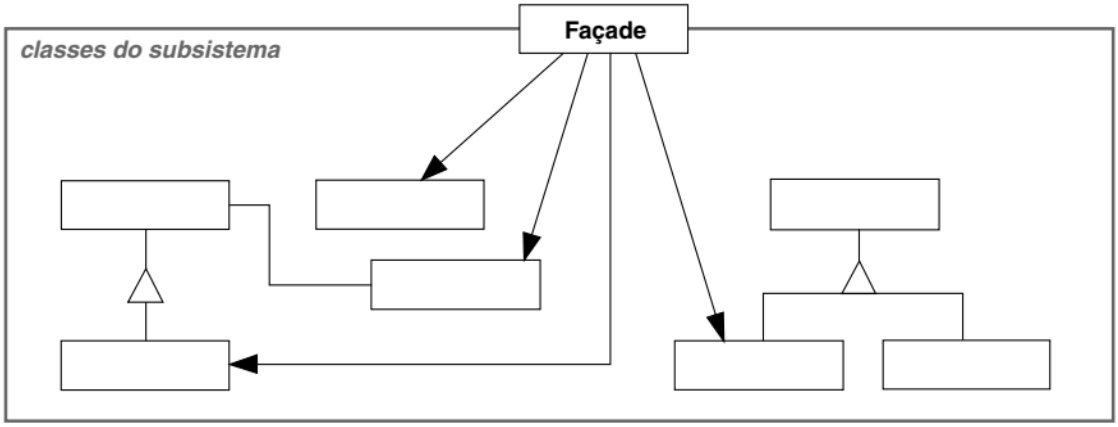
acessórios. Utilizar herança para representar todas as combinações é completamente inviável. Teríamos centenas de classes.

Com essa constatação, entra o padrão de projetos Decorator, criando os tipos de mudanças possíveis (atributos e comportamentos) como sendo objetos decorados bem como as armas que os anexam, e como objetos simples concretos as armas que não possuem acessórios. O método chave seria responsável por atualizar os atributos e comportamentos do objeto concreto (arma). Assim, permitindo que em tempo de execução seja aplicada a mudança no objeto arma, acrescentando ou removendo um dos efeitos dos acessórios.

4.2.5 Façade

O padrão de projeto *Façade* tem como intenção fornecer uma interface unificada para um conjunto de interfaces em um subsistema. O Façade define uma interface de nível superior que facilita o uso do subsistema. Encapsula um subsistema complicado, dando uma interface mais simples (GAMMA et al., 2007).

Diagrama 4-21 – Estrutura base do padrão de projetos *Façade*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Façade* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Compreensibilidade
- Estabilidade
- Manutenibilidade

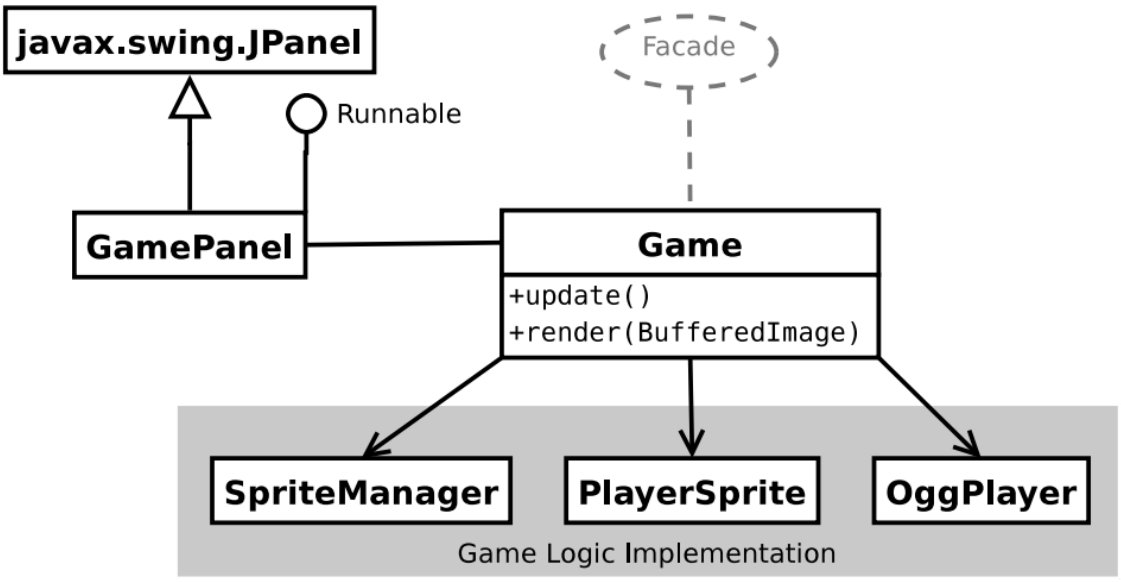
O padrão de projetos *Façade* possui as seguintes aplicabilidades (GAMMA et al., 2007):

- Fornecer uma interface simples para uma série de operações;
- Evitar alto acoplamento entre subsistemas devido às dependências entre eles;
- É desejado estruturar um subsistema em camadas, dando ao Façade o acesso de comunicação entre elas.

4.2.5.1 Uso em Jogos Digitais

Uma vez que o padrão de projetos *Façade* tem como uma das aplicabilidades evitar alto acoplamento, Gestwicki e Sun (2008) propõem o uso desse padrão para conectar de forma desacoplada o *model* de um jogo da programação de elementos visuais, considerando os diferentes tipos de resolução – tela cheia, janela etc. Assim, mesmo resoluções em mobile, desktop ou web podem ser tratadas sem afetar a lógica do jogo e vice-versa. Essa proposta é representada no Diagrama 4-23.

Diagrama 4-22 – *Façade* e desacoplamento.



Fonte: Gestwicki e Sun (2008). Demonstra a aplicação do padrão *Façade* para desacoplar a resolução do jogo de sua lógica de jogo.

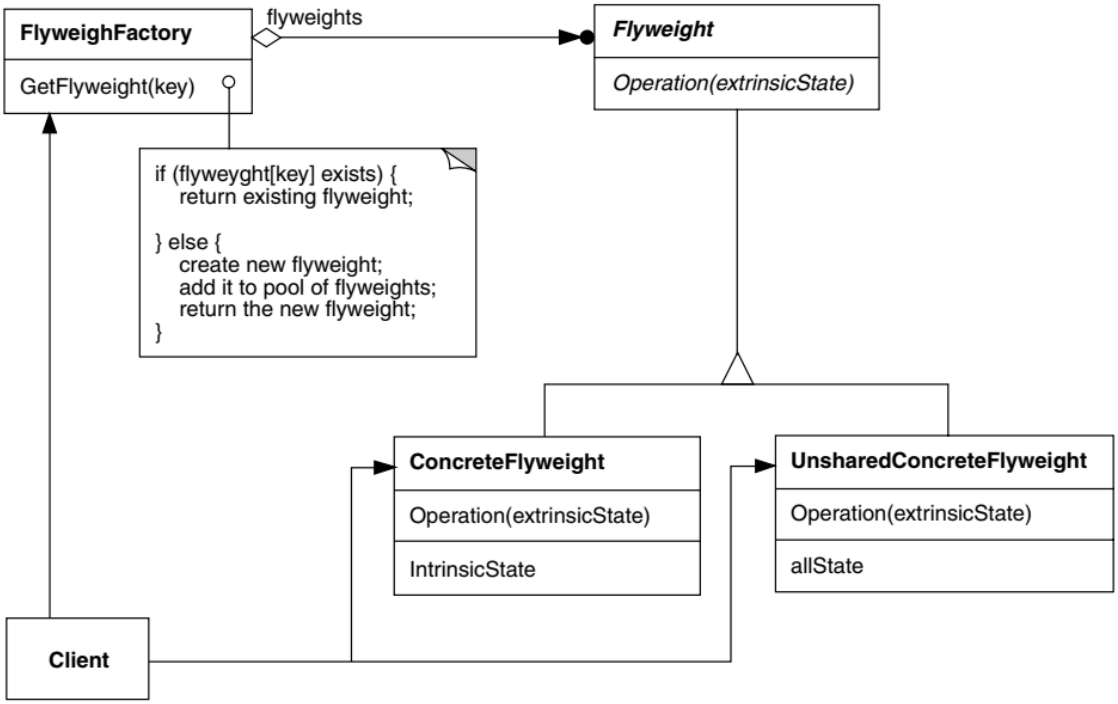
4.2.5.2 Equivalência com Jogo Digital Moderno

Muito comum na maioria dos jogos para desktop, é possível alterar a resolução do jogo para tela cheia, janela ou tela cheia em janela. Também é possível alterar a granularidade da tela, como “1920x1080” ou “1280x720” por exemplo. Essa funcionalidade é do mais alto nível de generalidade, pois pode pertencer a qualquer jogo digital. A título de especificação de comparativo, podemos indicar o jogo F1™ 2017 que possui essa funcionalidade.

4.2.6 Flyweight

O padrão de projetos *Flyweight* intenciona compartilhar o uso em larga escalada de objetos de granularidade fina (GAMMA et al., 2007). O objeto-*flyweight* é compartilhado de modo que pode ser usado em diferentes contextos simultaneamente.

Diagrama 4-23 – Estrutura base do padrão de projetos *Flyweight*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Flyweight* nenhum dos trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos a qualquer uma das propriedades descritos no trabalho.

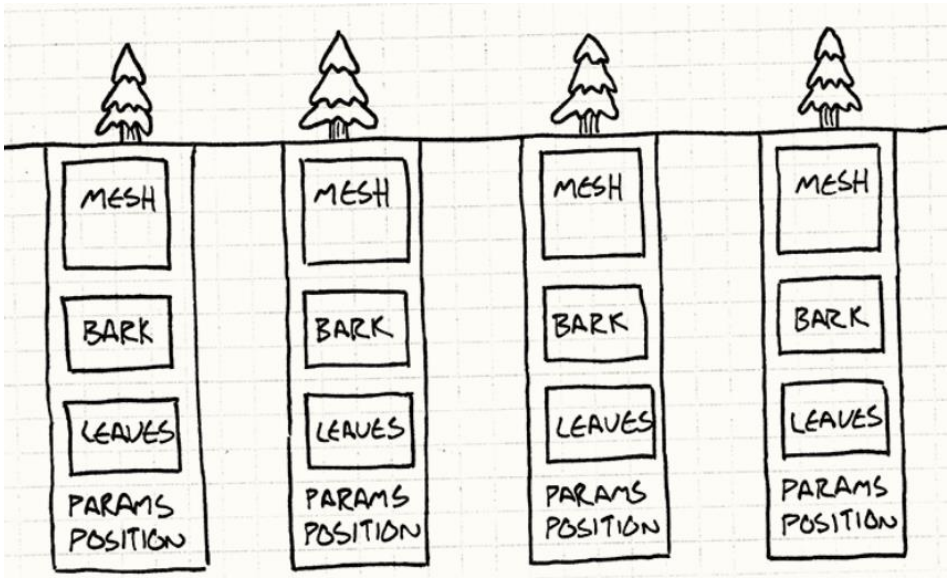
Diferente dos demais padrões, a lista de aplicabilidades do padrão de projetos *Flyweight* é de total dependência; segundo Erich Gamma (et al., 2007), todos as descrições devem ser verdadeiras para que a aplicabilidade dele seja eficaz:

- A aplicação faz uso de uma grande quantidade de objetos;
- A grande quantidade de objetos resulta em alto custo de armazenamento;
- A maioria dos estados do objeto pode ser ajustada para extrínseca (dependente do contexto externo ao objeto-*flyweight*);
- Objetos de um mesmo grupo podem ser substituídos por um grupo menor de objetos compartilhados sem os estados extrínsecos;
- A aplicação não depende da identidade desses objetos.

4.2.6.1 Uso em Jogos Digitais

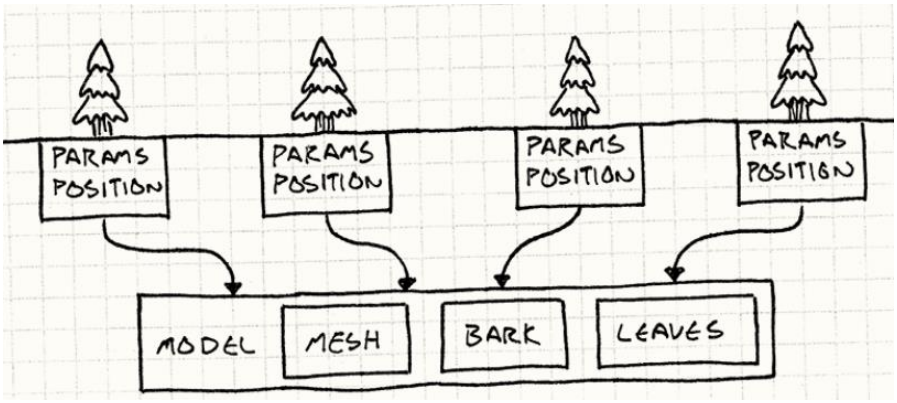
Dado o contexto de florestas em jogos, Nystrom (2014) argumenta que as árvores na maioria dos jogos digitais são um pequeno grupo variado que se repete em um cenário para formarem florestas ou simular a arborização de cidades, sendo essas arvores de decoração, sem um comportamento singular, ou seja, a identidade de cada arvore é irrelevante. Assim, Nystrom conclui que as arvores podem ser compartilhadas para construir esses cenários, poupando código, armazenamento e desempenho.

Figura 4-2 – Gerador de árvores sem *Flyweight*.



Fonte: Nystrom (2014). Estrutura de criação de árvores sem o padrão de projetos *Flyweight*. Cada arvore é tratada como tendo uma identidade singular. Ou seja, havendo 100 delas no cenário, serão 100 unidades processadas e armazenadas.

Figura 4-3 – Gerador de árvores com *Flyweight*.



Fonte: Nystrom (2014). Estrutura de criação de árvores com o padrão de projetos *Flyweight*. Há um grupo de árvores possíveis, com dados intrínsecos variáveis (model, mesh, bark, leaves) e dados extrínsecos variáveis (posições). Assim é possível criar uma variedade desejada de árvores cujo os objetos serão compartilhados, reaproveitando informações custosas ao armazenamento e desempenho.

Na Figura 4-3 fica claro o reaproveitamento de objetos-*flyweight*. Apenas os parâmetros de posicionamento das árvores não serão reaproveitados, sendo eles comumente um vetor 3D em jogos 3D ou um vetor 2D em jogos 2D. Todos os recursos mais custosos relacionados à renderização, são compartilhados.

4.2.6.2 Equivalência com Jogo Digital Moderno

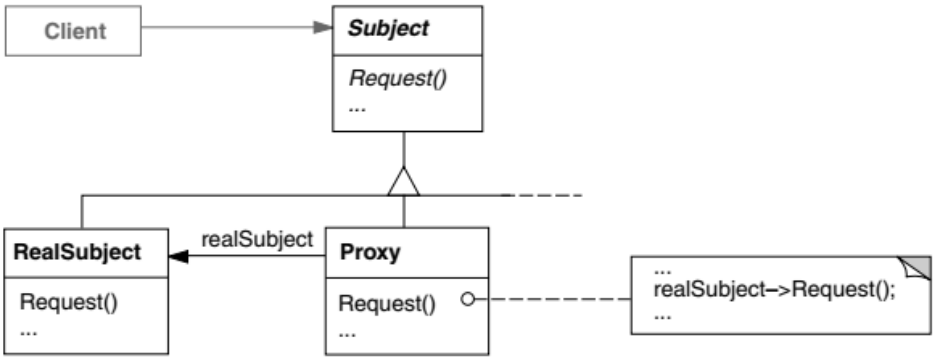
Flyweight é um padrão de projeto com grande aproveitamento no contexto dos jogos digitais. É facilmente inteligível que qualquer objeto decorativo de um cenário em jogo digital pode ser utilizado como objeto-*flyweight*.

Aprofundando-se no conceito do padrão de projeto *Flyweight*, pode-se citar os construtores de cenários, onde as partes do cenário como paredes, tochas, portas, janelas, telhados, estradas, entre outros, são objetos de um grupo com variedade limitada e cuja identidade deles não tem influência sobre o cenário.

4.2.7 Proxy

Também conhecido como *Surrogate, o* padrão de projeto *Proxy* intenciona fornecer um substituto ou marcador da localização de outro objeto para controlar o acesso a esse objeto (GAMMA et al., 2007). Ele adiciona um nível extra de acesso distribuído, controlado e inteligente que, através de um *wrapper*, evita que o componente real se torne muito complexo, poupando uma requisição de recursos que seriam subutilizados.

Diagrama 4-24 – Estrutura base do padrão de projetos *Proxy*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Proxy* nas propriedades:

- Adaptabilidade
- Manutenibilidade

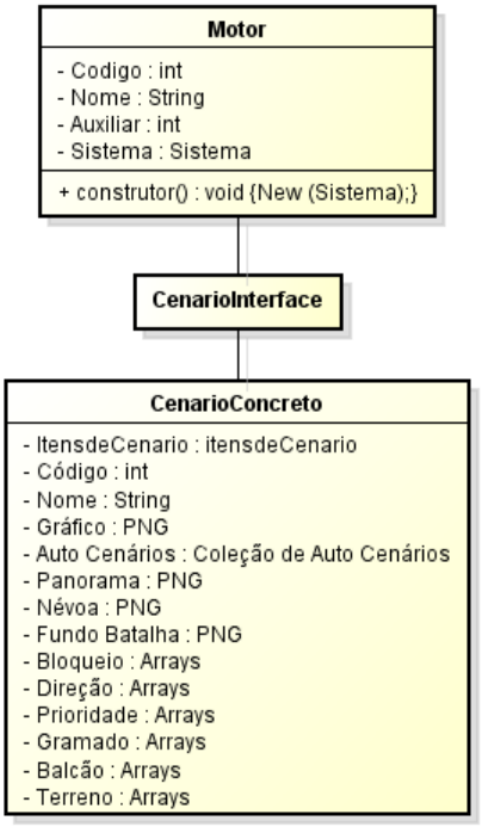
O padrão *Proxy* apresenta as seguintes aplicabilidades (GAMMA et al., 2007):

- É preciso um representante local para um objeto num espaço de endereçamento diferente;
- Criar objetos mais complexos sob demanda;
- Controlar o acesso ao objeto original, como um *wrapper*;
- Substituir um simples ponteiro por outro que executa ações adicionais quando um objeto, variável ou objeto é acessado.

4.2.7.1 Uso em Jogos Digitais

Apenas uma citação sobre o padrão de projetos *Proxy* foi encontrada no contexto de jogos digitais. O autor Figueiredo (2014) propõe, como demonstra o Diagrama 4-28, que o padrão Proxy é uma solução para manutenção e melhoria de desempenho na instanciação de cenários de jogos com múltiplos cenários.

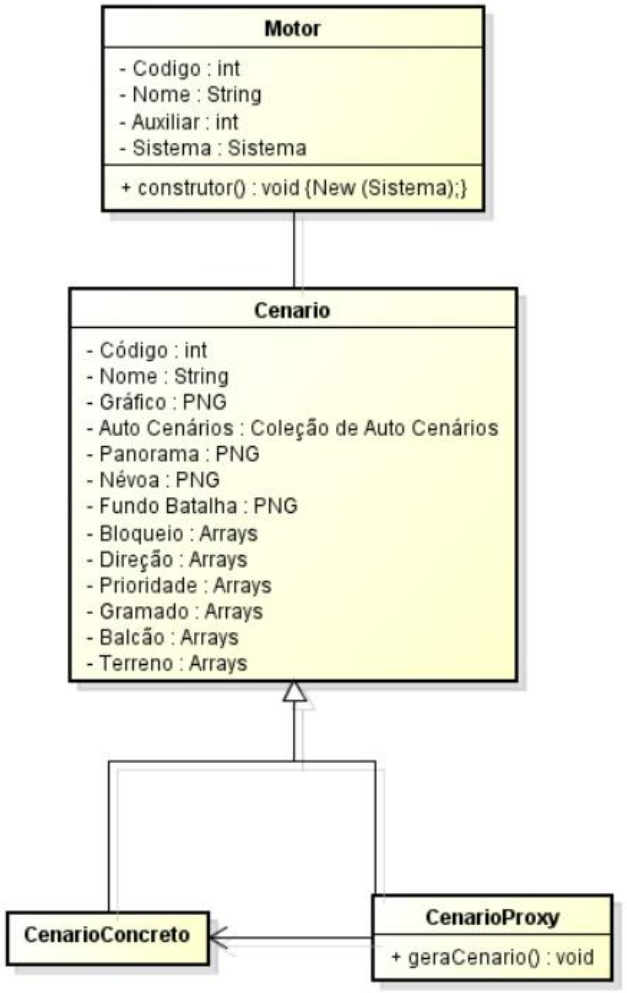
Diagrama 4-25 – Instanciação de cenários sem o padrão *Proxy*.



Fonte: Roberto Figueiredo (2014). Exemplo de diagrama para instanciação de cenários sem o padrão de projetos *Proxy*.

Ao alertar que, sendo necessário apenas um cenário por vez, sem o padrão o jogo precisa instanciar todos os elementos de todos os cenários no início do jogo, inviabilizando seu desempenho em muitas máquinas. Além disso, a manutenção e inclusão de novos cenários sem o padrão *Proxy* é onerosa pelo forte acoplamento da estrutura de classes. A solução proposta pelo autor é apresentada no Diagrama 4-29.

Diagrama 4-26 – Instanciação de cenários com o padrão *Proxy*.



Fonte: Roberto Figueiredo (2014). Diagrama para criação de cenários com o padrão de projetos *Proxy*.

4.2.7.2 Equivalência com Jogo Digital Moderno

A proposta do autor Figueiredo é não apenas contundente, como também atende à uma funcionalidade bastante genérica e utilizada nos jogos digitais, tanto em jogos de décadas atrás quanto nos modernos, a exemplo do jogo *Player's Unknows Battlegrounds* (2017), em que 100 personagens de jogadores se movimentam continuamente ao longo de partida *multiplayer*, de maneira que todos os clientes conectados devem ser informados dessas movimentações. Todavia, é relevante que ao atualizar a posição dos demais personagens que não estejam ao alcance do jogador, esses personagens não precisem utilizar recursos de renderização, animação ou sons, uma vez que os resultados desses recursos não

serão aplicados ao jogador por não os visualizar ou tem o alcance mínimo para os ouvir.

4.3 PADRÕES COMPORTAMENTAIS

A categoria de padrões de projetos comportamentais tem como característica a abordagem sobre a comunicação entre os objetos. Os padrões comportamentais de classe distribuem os comportamentos entre as classes através de herança. Os padrões comportamentais de objetos substituem herança pela composição de objetos (GAMMA et al., 2007). Os trabalhos que utilizam padrões de projetos comportamentais para solucionar funcionalidades de jogos digitais são apresentados na tabela 6.

Tabela 6 – Trabalhos inclusos sobre padrões comportamentais

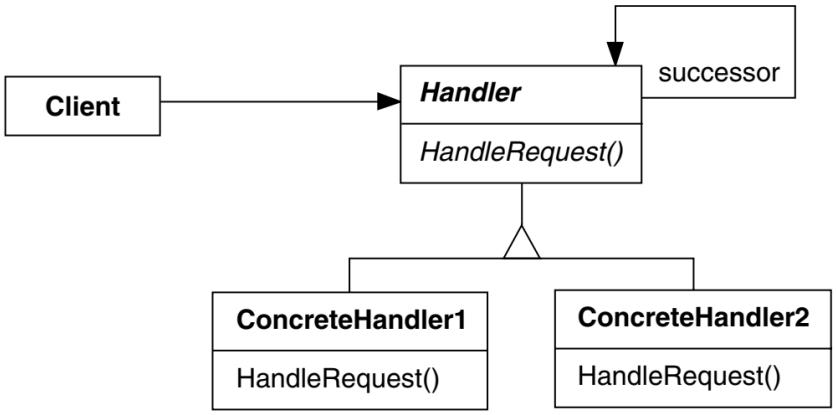
Padrão	**IDs**
Chain of Responsibility	TL_12
Command	TL_03, TL_04, TL_05, TL_07, TL_08, TL_12, TL_15
Interpreter	TL_12
Iterator	TL_07, TL_12
Mediator	TL_02, TL_04, TL_12
Memento	TL_07, TL_10, TL_12
Observer	TL_06, TL_07, TL_08, TL_09, TL_12, TL_15, TL_16, TL_17
State	TL_01, TL_03, TL_04, TL_05, TL_06, TL_07, TL_08, TL_09, TL_10, TL_12, TL_15, TL_16, TL_20
Strategy	TL_01, TL_04, TL_05, TL_06, TL_07, TL_09, TL_10, TL_11, TL_12, TL_16, TL_17, TL_19
Template Method	TL_09, TL_11, TL_12, TL_17
Visitor	TL_03, TL_05, TL_09, TL_12, TL_15, TL_16

Fonte: Jadson Almeida (2018).

4.3.1 *Chain of Responsibility*

O padrão de projetos *Chain of Responsibility* intenciona evitar o acoplamento de uma cadeia de tratamentos em só objeto. *Chain of Responsibility* é um padrão que segrega um conjunto de responsabilidades de tratamento de solicitação, distribuindo os tipos de tratamento em diferentes objetos (GAMMA et al., 2007).

Diagrama 4-27 – Estrutura base do padrão de projetos Chain of *Responsibility*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Chain of Responsibility* nas propriedades:

- Adaptabilidade
- Compreensibilidade
- Manutenibilidade

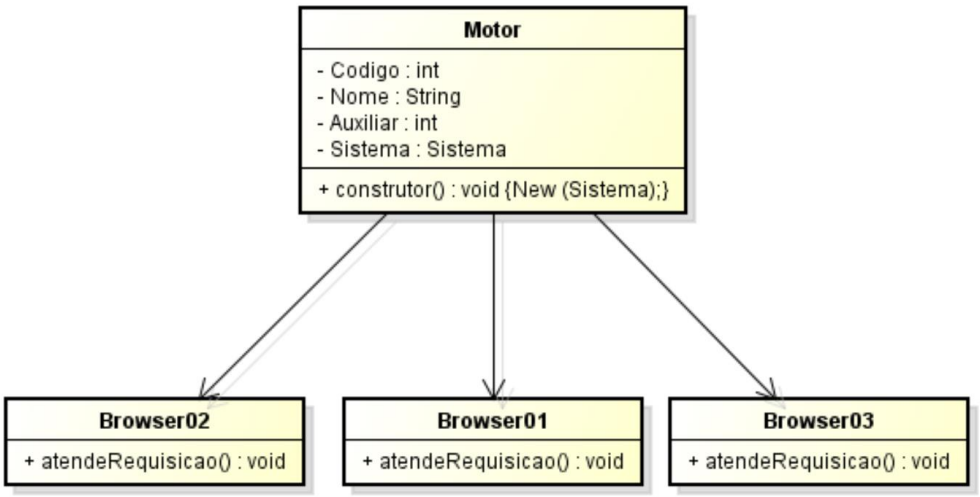
Para Erich Gamma (et al., 2007) o padrão de projetos *Chain of Responsibility* possui as seguintes aplicabilidades:

- Um objeto deve receber mais de um tratamento e o objeto responsável pelo tratamento, além de não conhecer o objeto a ser tratado, também precisa ser escolhido automaticamente;
- Dentre um conjunto de objetos, é preciso escolher um para enviar uma solicitação sem especificar o receptor;
- A solicitação precisa ser tratada por um conjunto de objetos selecionados dinamicamente.

4.3.1.1. Uso em Jogos Digitais

O autor Roberto Figueiredo cita um problema comum em aplicações para plataforma WEB, o qual jogos digitais também estão inclusos. Os diferentes navegadores trazem, cada qual, uma lista diferente de suportes tecnológicos que acarretam em um comportamento diferente, muitas vezes indesejado, como uma má resolução do jogo no navegador, algum elemento gráfico apresentado de forma errada ou mesmo o não funcionamento do jogo. Além dos diferentes navegadores, eles também são divididos entre várias versões, aumentando a complexidade de tratamento desse cenário. No Diagrama 4-31 Figueiredo apresenta uma forma genérica de tratamento ao problema sem o uso de padrões de projeto.

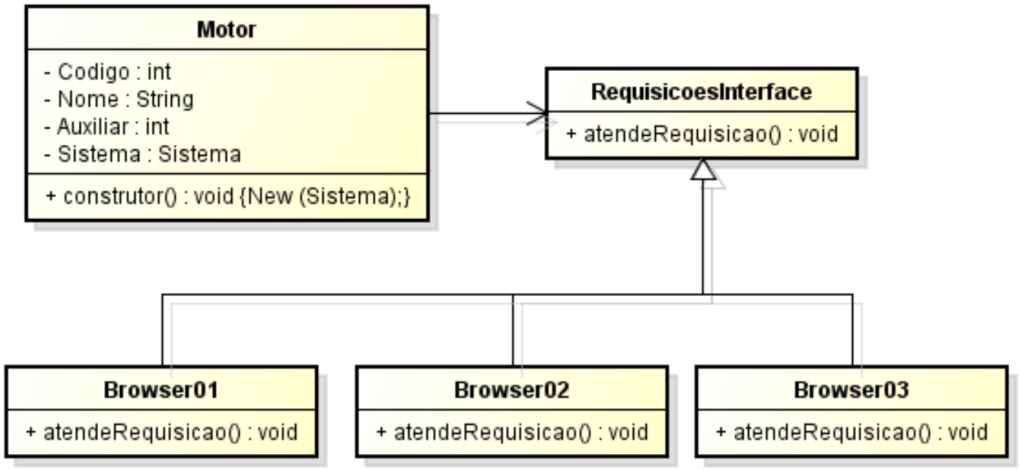
Diagrama 4-28 – Tratamento de navegadores sem o padrão *Chain of Responsibility*.



Fonte: Roberto Figueiredo (2014). Estrutura genérica para tratamento de jogos digitais em diferentes navegadores, sem o uso de padrões de projeto.

Como solução, Figueiredo propõe a aplicação do padrão de projetos *Chain of Responsibility*, criando, como demonstra o Diagrama 4-32, uma interface em comum com todos os objetos que tratam a compatibilidade do jogo com os diferentes navegadores, sendo possível, através desse desacoplamento, remover e criar novos tratamentos sem que seja necessário alterar os já existentes ou o próprio motor do jogo.

Diagrama 4-29 – Tratamento de navegadores com o padrão *Chain of Responsibility*.



Fonte: Roberto Figueiredo (2014). Estrutura criada para tratamento de jogos digitais em diferentes navegadores, utilizando o padrão de projeto *Chain of Responsibility*. A funcionalidade de tratamento é desacoplada do motor do jogo e atribuída a uma série de objetos que, de maneira encadeada e automática, será selecionado o objeto correspondente ao navegador utilizado pelo jogador.

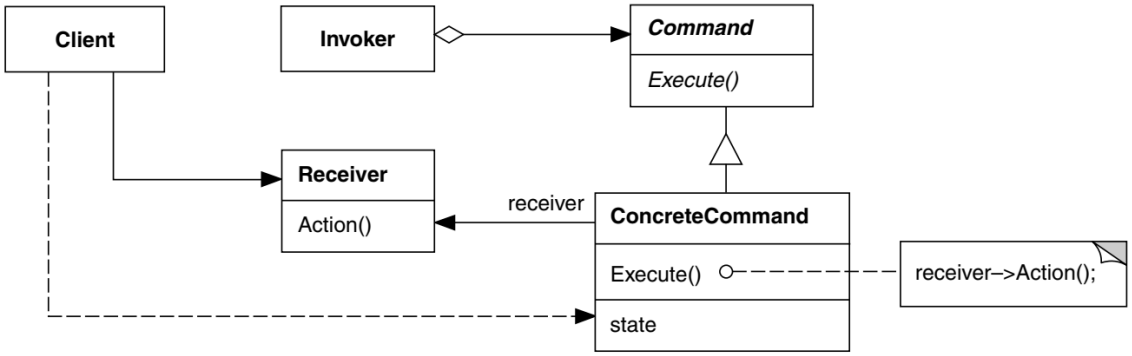
4.3.1.2. Equivalência com Jogo Digital Moderno

Qualquer jogo que use como plataforma de execução um navegador, pode ser beneficiado com o uso do padrão de projetos *Chain of Responsibility*, como demonstrado por Roberto Figueiredo (2014). A eficácia da estrutura de tratamento dada pelo padrão independe dos demais detalhes técnicos do jogo, como controles, gráfico, som etc.

4.3.2 *Command*

O padrão de projetos *Command* intenciona transformar uma solicitação em um objeto encapsulado, parametrizando clientes com diferentes solicitações e criando um registro de solicitações de modo que elas possam ser desfeitas e refeitas (GAMMA et al., 2007).

Diagrama 4-30 – Estrutura base do padrão de projetos *Composite*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Command* nas propriedades:

- Estabilidade
- Manutenibilidade

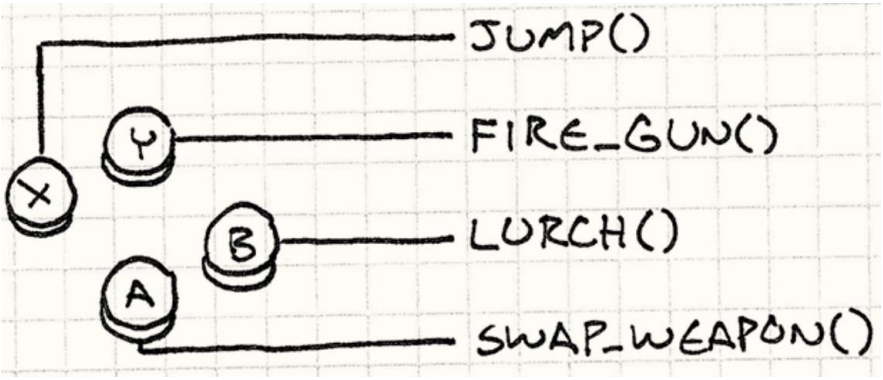
O autor Erich Gamma (et al., 2007) atribui ao padrão de projetos *Command* as seguintes aplicabilidades:

- É desejável parametrizar objetos pela da execução de uma ação;
- É importante atribuir orientação a objetos para o uso de *callbacks*;
- As solicitações precisam ser especificadas, enfileiradas e executadas em tempos diferentes;
- As solicitações precisam suportar as operações de desfazer e refazer;
- O sistema precisa ser estruturado por operações de alto nível que são compostas por operações primitivas.

4.3.1.3. Uso em Jogos Digitais

Uma vez que o padrão de projetos Command tem como uma de suas aplicabilidades o encapsulamento de solicitações, Robert Nystrom (2014) utiliza esse padrão para estruturar o controle de comandos do jogador. Dado os comandos como mostra a Figura 4-4, o jogador precisa com o botão “X” pular, com o botão “Y” atirar, com o botão “B” agachar e o botão “A” trocar de arma.

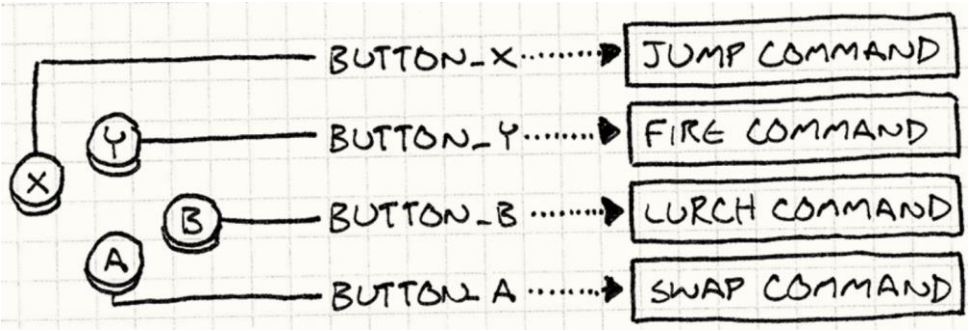
Figura 4-4 – Controle do jogador sem o padrão *Command*.



Fonte: Robert Nystrom (2014). Estrutura de comandos do jogador (saltar, atirar, agachar e trocar arma), necessários para implementação.

Com a aplicação do padrão de projetos *Command*, cada ação passa a ser encapsulada e com isso o jogador pode configurar os botões do controle, pois como demonstra a Figura 4-5, o botão agora aciona um objeto e pode ser configurado para acionar outro objeto ao invés do *default*.

Figura 4-5 – Controle do jogador com o padrão *Command*.



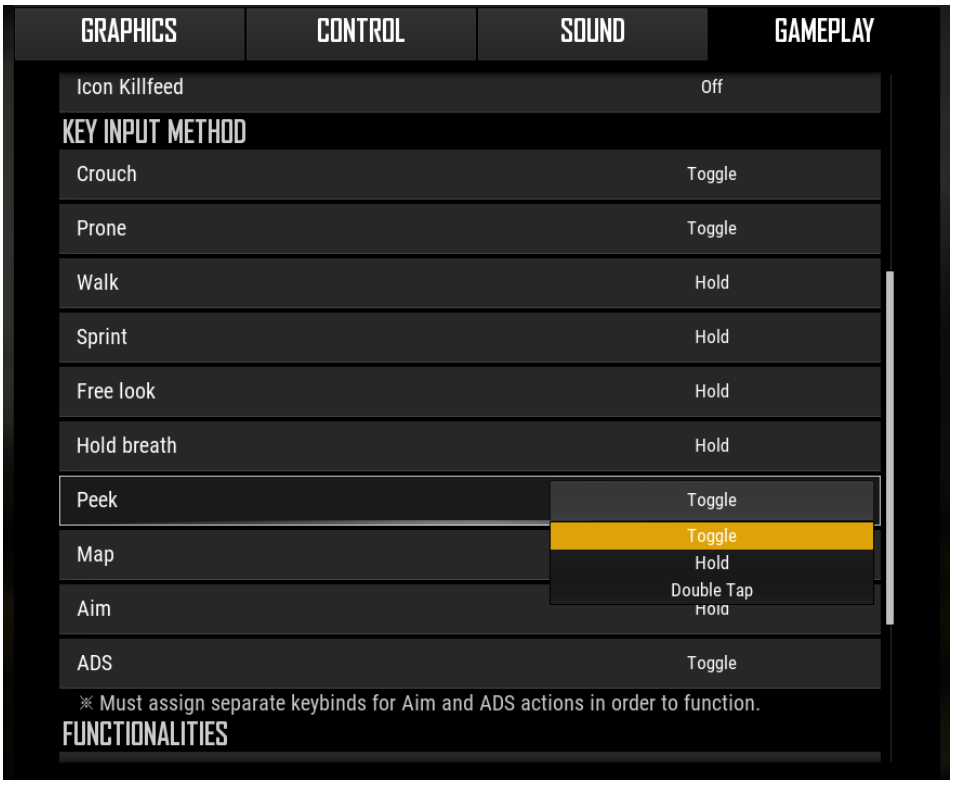
Fonte: Robert Nystrom (2014). Nova estrutura de comandos do jogador, agora com o padrão de projetos *Command* encapsulando cada ação.

Além do jogo ter agora suporte para a criação de comandos/ações independentes, o jogo poderia facilmente dar suporte aos comandos *Fire* e *Lurch* se tornarem *toggles*. O jogador clica uma vez no botão do comando *Fire*, por exemplo, e ele atira initerruptamente até o jogador clicar novamente, desfazendo a ação; o mesmo com o comando *Lurch* para agachar e ficar de pé.

4.3.1.4. Equivalência com Jogo Digital Moderno

Qualquer jogo que possua opções de desfazer e refazer uma ação, como comandos de *toggle*, podem se beneficiar do padrão de projetos *Command*. No jogo *Player's Unknows Battlegrounds* é possível que alguns comandos sejam desativados apenas soltando o botão ou por clique (*toggle*), como é o caso dos comandos de espiar, correr e falar.

Figura 4-6 – Opções de controle do jogo Player's Unknows Battlegrounds.

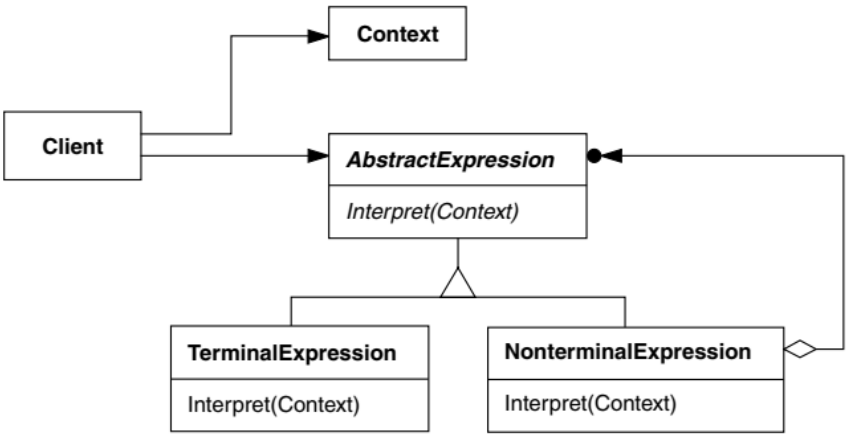


Fonte: Jadson Almeida (2018). Captura de tela da janela de configuração de jogabilidade do jogo *Player's Unknows Battlegrounds*, com as opções de *toggle* (alternância), *hold* (segurar) e clique duplo (*double tap*) para ativação/desativação de uma ação do personagem.

4.3.3 *Interpreter*

O padrão de projetos *Interpreter* intenciona definir uma representação para a gramática de uma linguagem, onde um interpretador traduz as sentenças da linguagem utilizando a representação da gramática (GAMMA et al., 2007).

Diagrama 4-31 – Estrutura base do padrão de projetos *Interpreter*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Interpreter* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Compreensibilidade
- Manutenibilidade

Para Erich Gamma (et al., 2007), o padrão de projetos *Interpreter* faz sentido no contexto onde há uma linguagem a ser interpretada e é possível representar as sentenças da linguagem através de árvores sintáticas abstratas. Nesse contexto, os autores atribuem ao *Interpreter* as seguintes aplicabilidades:

- A gramática da linguagem é simples;
- A eficiência da tradução não é crítica.

Sobre a eficiência da tradução, os autores não apresentam uma argumentação concreta, descrevem apenas que os interpretadores mais eficientes não implementam árvores de análise sintática, e apenas citam que máquinas de estado são frequentemente usadas em expressões regulares. O autor Freeman (et al., 2007) também afirma que há um problema de eficiência no padrão de projetos *Interpreter*, mas igualmente não apresenta argumentações ou exemplos justificativos.

4.3.1.5. Uso em Jogos Digitais

O único autor dos trabalhos levantados, a trazer o padrão de projetos Interpreter para o contexto dos jogos digitais, foi Figueiredo (2014). O autor afirma que em sua pesquisa que não encontrou trabalhos relacionando o *Interpreter* à jogos digitais. Todavia, propõe o uso do *Interpreter* para jogos simuladores de perguntas e respostas, onde as respostas não seriam objetivas, mas sim discursivas.

Apesar da proposta de Figueiredo, o próprio conclui que a alta complexidade em analisar respostas discursivas é algo a ser pesquisado. Essa constatação contradiz a aplicabilidade restritiva do Interpreter a gramáticas simples. Assim, não há nos trabalhos analisados, uma aplicação do padrão de projetos Interpreter solucionando uma funcionalidade de jogo digital cuja justificativa seja plausível à própria definição do padrão.

4.3.1.6. Equivalência com Jogo Digital Moderno

Apesar de nenhum dos 12 trabalhos analisados conter uma aplicação plausível do *Interpreter* em jogos digitais, o projeto L2J, que é uma versão na linguagem JAVA para servidores do *MMORPG Lineage* 2, possui uma funcionalidade que se enquadra perfeitamente à aplicabilidade do padrão. O L2J, para controlar as funcionalidades relacionadas à *network*, utiliza códigos em hexadecimal para realizar a comunicação entre cliente e servidor; tanto o envio de dados do cliente para o servidor quanto o envio de dados do servidor para o cliente são dados em hexadecimal que representam ações e resultados, como a solicitação do jogador para que o personagem se movimente ou ataque, e, a resposta do servidor à essa solicitação, como mover ou não o personagem (mediante regras) e o resultado do ataque do personagem (se foi bem sucedido ou não) etc.

No caso do L2J, os códigos hexadecimais são convertidos em comandos e os esses comandos acionam ações. Para isso, há uma classe abstrata para receber comandos e outra para enviá-los; cada comando em si herda dessa classe abstrata e faz o tratamento pertinente. Se aplicarmos o padrão de projetos *Interpreter*, podemos ter uma combinação de comandos que formam um comando final (*NonterminalExpression*), como é o caso do ataque contra outro personagem (*PvP*). No combate contra monstros controlados pela inteligência artificial (*PvE*) o jogador precisa apenas clicar com o botão direito do mouse contra o monstro. Mas no

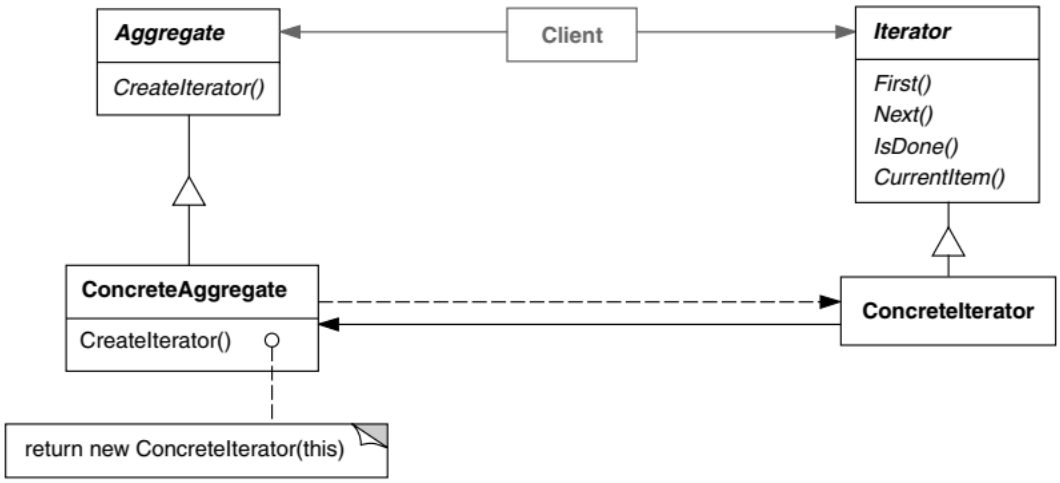
combate personagem contra personagem (*PvP*), o jogador precisa segurar a tecla CTRL do teclado e, enquanto segura a tecla, clicar com o botão direito do mouse no personagem alvo do ataque. Há uma série de outras combinações de comandos que podem ser tratadas com o padrão de projetos *Interpreter* (*NonterminalExpression*).

Entretanto, apesar da funcionalidade enquadrar-se na aplicabilidade do padrão Interpreter, ela não é uma equivalência com nenhuma outra identificada nos trabalhos incluídos. Mas sim uma proposta de solução. Importante frisar que não foi validada além do contexto conceitual.

4.3.4 *Iterator*

Os autores Gamma et al. (2007) define o padrão de projetos Iterator como um fornecedor de acesso sequencial à uma lista de elementos sem expor a representação subjacente do objeto a que eles pertencem. Há linguagens que oferecem uma interface nativa para a implementação do Iterator, como JAVA e C#.

Diagrama 4-32 – Estrutura base do padrão de projetos *Iterator*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Iterator* nas propriedades:

- Adaptabilidade
- Compreensibilidade
- Manutenibilidade

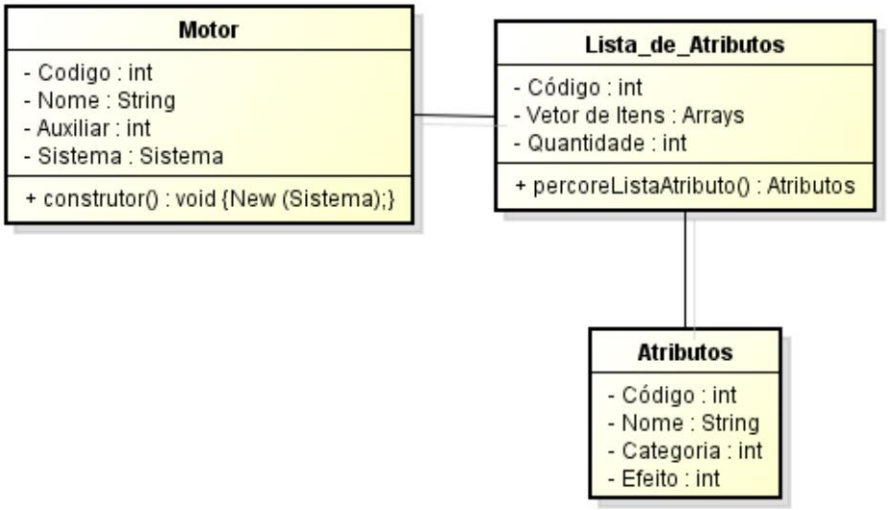
O padrão de projetos *Iterator* possui as seguintes aplicabilidades (GAMMA et al., 2007):

- É desejável acessar objetos agregados independente de sua representação interna;
- É preciso que uma lista de objetos suporte múltiplos percursos;
- É preciso fornecer uma interface uniforme que dê suporte à iteração polimórfica.

4.3.1.7. Uso em Jogos Digitais

Assim como o padrão *Interpreter*, o único autor que abordou uma aplicação do padrão de projeto Iterator solucionando uma funcionalidade de jogo digital, foi Figueiredo (2014). O autor cita o jogo “Os Federa 4 e a Máquina do Tempo”, no qual há a funcionalidade de evolução dos personagens, que, ao adquirirem um nível maior, os atributos do personagem são alterados.

Diagrama 4-33 – Evolução de personagens sem o padrão *Iterator*.

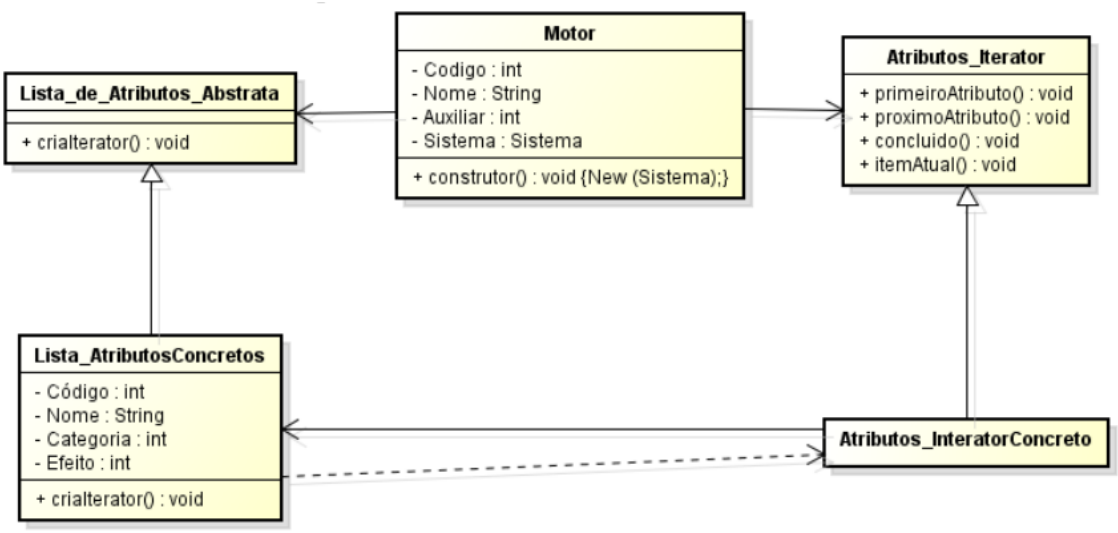


Fonte: Roberto Figueiredo (2014). Proposta de solução para evolução de personagens no jogo “Os Federa 4 e a Máquina do Tempo”, sem o uso de padrões de projeto.

A proposta de Figueiredo consiste em criar um agregado de elementos para cada atributo, com índice correspondente ao nível do personagem, e, com valor correspondente ao valor do atributo do personagem. Assim, o motor de jogo, a lista

de atributos e a busca nessa lista estão todos desacoplados entre si, como demonstra no Diagrama 4-39.

Diagrama 4-34 – Evolução de personagens com o padrão *Iterator*.



Fonte: Roberto Figueiredo (2014). Proposta de solução para evolução de personagens no jogo “Os Federa 4 e a Máquina do Tempo”, utilizando o padrão de projetos Iterator. Com o *Iterator*, o motor de jogo, a lista de atributos do personagem e a busca por esses atributos estão todos desacoplados entre si.

4.3.1.8. Comparativo com Jogos Digitais

No jogo *Foxhole* há 20 tipos de patentes, como “Soldado” e “Capitão”, que o personagem do jogador pode ter. O ranking de patentes depende da quantidade de pontos de recomendação que o personagem possui. Para cada um dos 20 níveis de ranking, há uma determinada quantidade de pontos que devem ser alcançados.

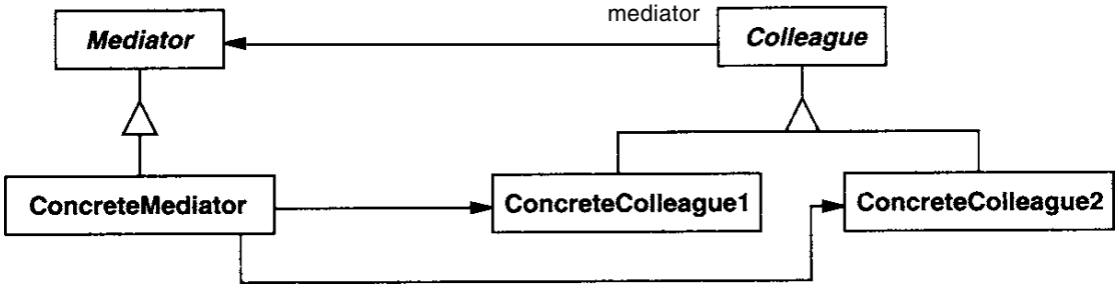
Assim como em “Os Federa 4 e a Máquina do Tempo”, onde há uma lista de valores a qual interessa as funções do *Iterator* “*primeiroAtributo*”, “*proximoAtributo*” e “itemAtual”, a mesma estrutura pode ser usada para percorrer os níveis da patente do *Foxhole* como “primeiraPatente”, “proximaPatente” e “patenteAtual”.

4.3.5 *Mediator*

O padrão de projetos *Mediator* intenciona criar um objeto que encapsula a forma de interação entre um conjunto de objetos. Ele promove o baixo acoplamento

entre os objetos e permite que as interações entre eles variem independentemente (GAMMA et al., 2007).

Diagrama 4-35 – Estrutura base do padrão de projetos *Mediator*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *Mediator* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Acoplamento
- Compreensibilidade
- Manutenibilidade

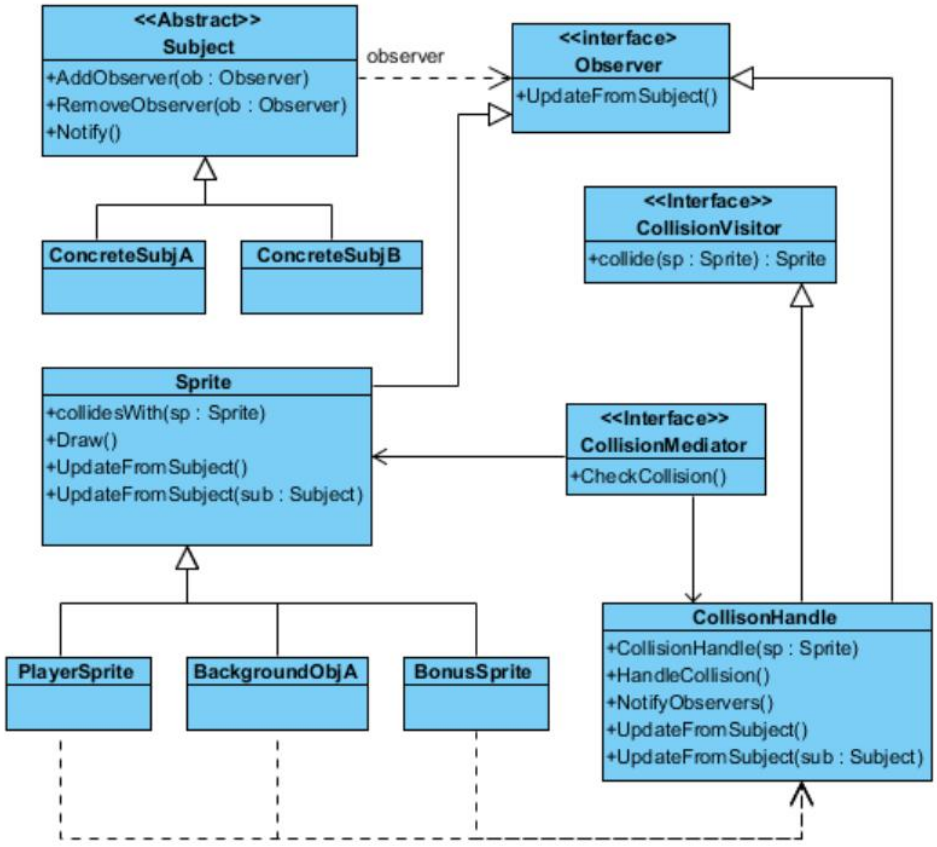
Para Erich Gamma (et al., 2007), o padrão de projetos Mediator possui as seguintes aplicabilidades:

- A comunicação entre um conjunto de objetos, apesar de bem definida, é também complexa;
- É desejável reutilizar objetos, mas eles utilizam muitas referências de outros objetos;
- Vários objetos são responsáveis por um comportamento, e, este deveria ser customizável ou adaptável sem o uso excessivo de subclasses.

4.3.1.9. Uso em Jogos Digitais

Junfeng Qu, Yinglei Song e Yong Wei (2013) trazem diversas funcionalidades implementando padrões de projeto. Entre elas, o controle de colisões entre *Sprites* (AMIGA DEVELOPER, 1989) em um jogo 2D.

Diagrama 4-36 – Controle de colisões com *Mediator*, *Observer* e *Visitor*.



Fonte: Junfeng Qu, Yinglei Song e Yong Wei (2013). Estrutura para o controle de colisões de sprites em um jogo 2D. Nela temos os padrões de projeto *Observer*, *Visitor* e *Mediator*.

Para evitar o acoplamento entre as sprites do jogador, do cenário e de objetos gerais (armas, munições, explosões etc.), o padrão de projetos *Mediator* é implementado para fazer a comunicação entre esses objetos, tornando-os mais independentes.

4.3.1.10. Equivalência com Jogo Digital Moderno

Apesar da aplicação do padrão de projetos *Mediator*, pelo autor Junfeng (et al., 2013), em um jogo 2D, a mesma aplicação pode ser feita em jogos 3D que possuírem um subsistema de colisões, como é o caso do jogo *SpellForce 3*, em que vários personagens são trabalhados pelo motor de colisão, juntamente com o cenário e demais elementos colidíveis.

Figura 4-7 – Campo de batalha do jogo SpellForce 3.

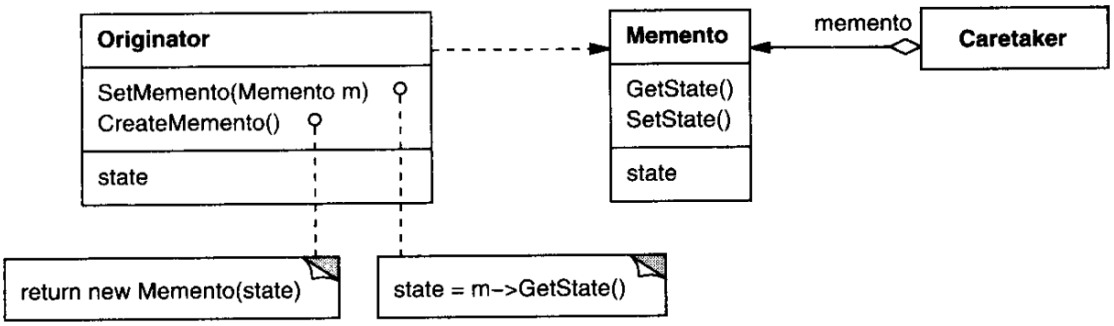
Fonte: Jadson Almeida (2018). Campo de batalha do jogo *SpellForce*, com vários elementos colidíveis a serem manipulados.

A Figura 4-7 demonstra a alta quantidade de elementos colidíveis a serem manipulados pelo motor de colisões, como as dezenas de personagens na imagem, o fogo que precisa colidir contra eles para surtir efeito e o próprio cenário. Com o *Mediator*, é possível evitar o alto acoplamento entre todos esses elementos.

4.3.6 *Memento*

O padrão de projetos *Memento*, também conhecido como *Token*, intenciona capturar e externalizar um estado de um objeto para que esse estado possa ser restaurado em algum momento sem violar o encapsulamento (GAMMA et al., 2007).

Diagrama 4-37 – Estrutura base do padrão de projetos *Memento*.



Fonte: Erich Gamma (et al., 2007)

Nenhum dos trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos a quaisquer das propriedades nele descritas para o padrão de projetos *Memento*.

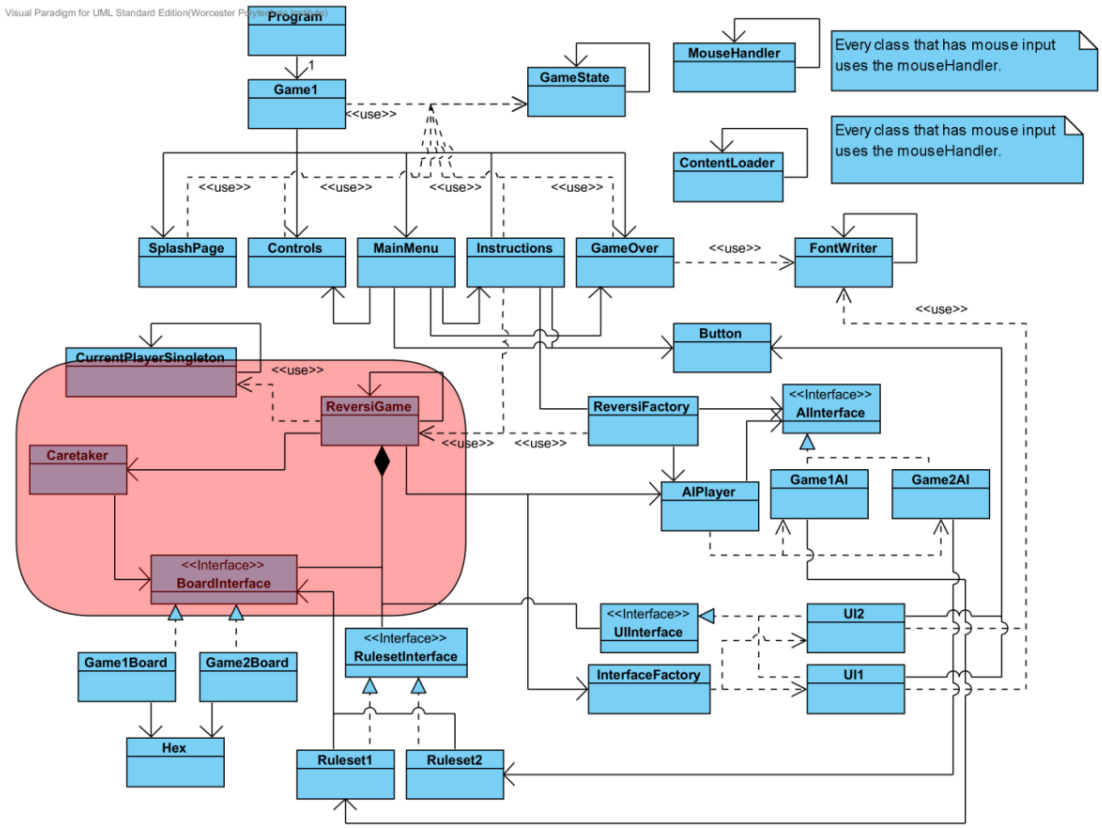
Segundo Erich Gamma (et al., 2007) o padrão de projetos *Memento* possui as seguintes aplicabilidades:

- O estado de um objeto deve ser salvo para que possa ser restaurado depois;
- Uma interface usada diretamente para obter o estado do objeto violaria o encapsulamento.

4.3.1.11. Uso em Jogos Digitais

Reversi é um jogo de tabuleiro semelhante à Dama, também para dois jogadores. Numa versão digital, para abordar a funcionalidade de salvar o jogo, Zawada (2013) utiliza o padrão de projetos *Memento*.

Diagrama 4-38 – Reversi com o padrão *Memento*.



Fonte: Zawada (2013). Estrutura da versão do jogo *Reversi*. Os elementos dentro do retângulo avermelhado (*Caretaker*, *ReversiGame* e *BoardInterface*) compõem a implementação do padrão *Memento*.

A classe *ReversiGame* – que é um controlador do jogo – é um *Singleton* que contém uma instância de *BoardInterface* (*Game1Board* ou *Game2Board*) – essa interface representa o comportamento do tabuleiro. No início da partida o *ReversiGame* passa a instância atual do *BoardInterface* para o *Caretaker* e assim pode solicitar a ele o estado do *BoardInterface* em qualquer ponto posterior.

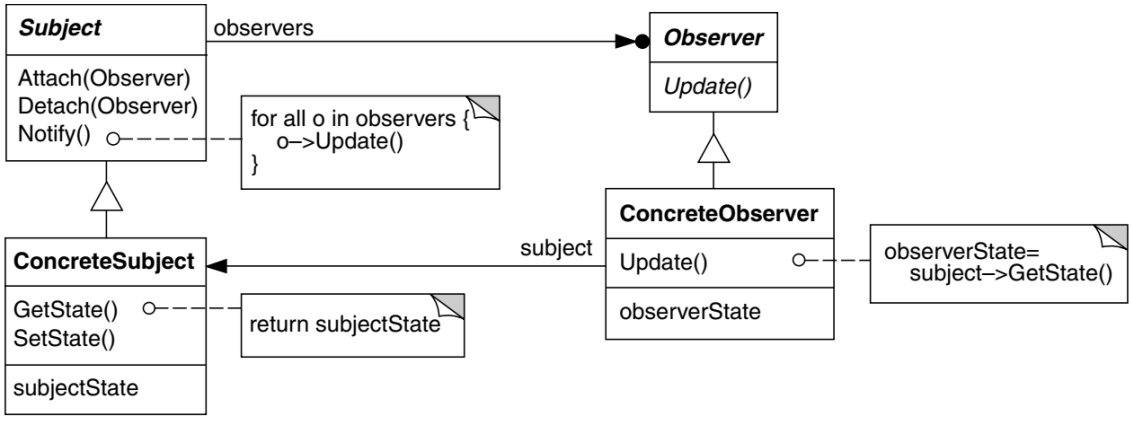
4.3.1.12. Comparativo com Jogo Digital Atual

Muitos jogos possuem a funcionalidade de salvar o estado atual, sendo possível carrega-lo posteriormente. É o caso de *SpellForce 3* – onde é possível retomar a partida jogada em qualquer momento, restaurando tudo que estava acontecendo no instante do salvamento; também é presente em *Pro Evolution Soccer 2018* e *F1™ 2017* – onde os jogadores podem salvar o estado dos campeonatos (como mundiais) e das campanhas do esporte (como o modo carreira).

4.3.7 *Observer*

A intenção do padrão de projetos *Observer* é atribuir uma dependência entre objetos onde a mudança de estado de um é notificada nos objetos dependentes que, por sua vez, podem ser atualizados com base na notificação recebida. Essa dependência é dita um-para-muitos (GAMMA et al., 2007).

Diagrama 4-39 – Estrutura base do padrão de projetos *Observer*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Observer* nas propriedades:

- Acoplamento
- Adaptabilidade
- Manutenibilidade
- Polimorfismo

O padrão de projetos Observer possui as seguintes aplicabilidades (GAMMA et al., 2007):

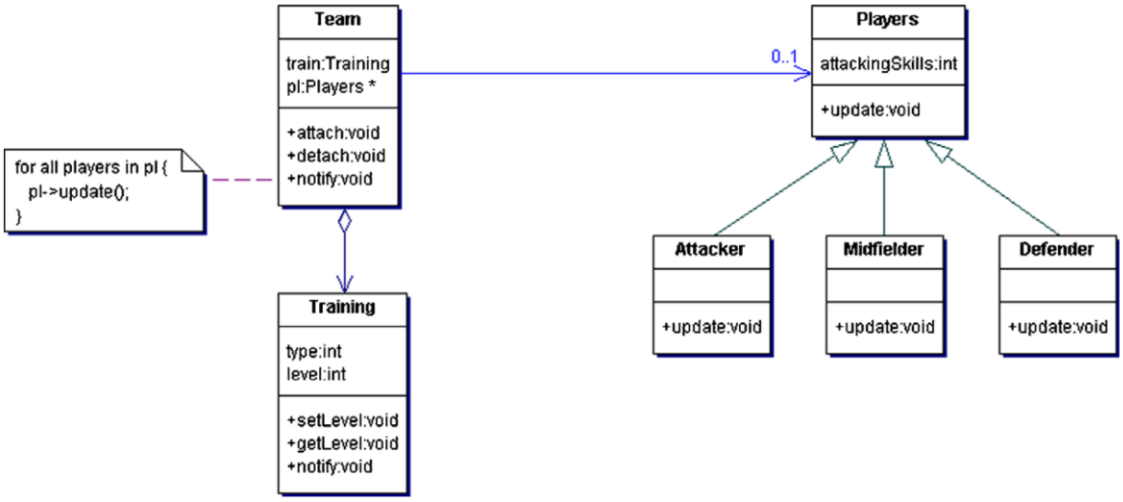
- É desejável encapsular os aspectos de uma abstração de objeto, para que possam ser reutilizadas independentemente;
- Há objetos que são dependentes da mudança de estado de outro objeto e não é previsto quantos objetos são dependentes;
- É desejável evitar acoplamento entre objetos notificadores e objetos que recebem a notificação.

4.3.1.13. Uso em Jogos Digitais

O padrão de projetos *Observer* é usado em jogos ditos exigentes, segundo Ampatzoglou e Chatzigeorgiou (2006), que aplicam o padrão no contexto de um jogo de gerenciamento de futebol, onde jogadores do time participam de um centro de treinamento que, com o tempo, este centro de treinamento vai avançar de nível,

melhorando as habilidades dos jogadores que estão treinando nele. Quando o centro de treinamento avançada de nível, todos os jogadores que treinam nele são notificados do avançado, e assim podem atualizar seus valores referentes ao treinamento que faziam. Um jogador que treinava cobrança de pênalti, elevará seu valor de cobrança de pênalti quando o centro de treinamento avançar de nível.

Diagrama 4-40 – Gerenciador jogadores de futebol com o padrão *Observer*.



Fonte: Ampatzoglou e Chatzigeorgiou (2006). Aplicação do padrão de projetos *Observer* em um jogo de gerenciamento de futebol, onde o centro de treinamento (*Training*), ao avançar de nível (*level*), notifica os jogadores (*Players*) que estão treinando nele, e, esses jogadores atualizam suas habilidades (*attackingSkills*).

4.3.1.14. Equivalência com Jogo Digital Moderno

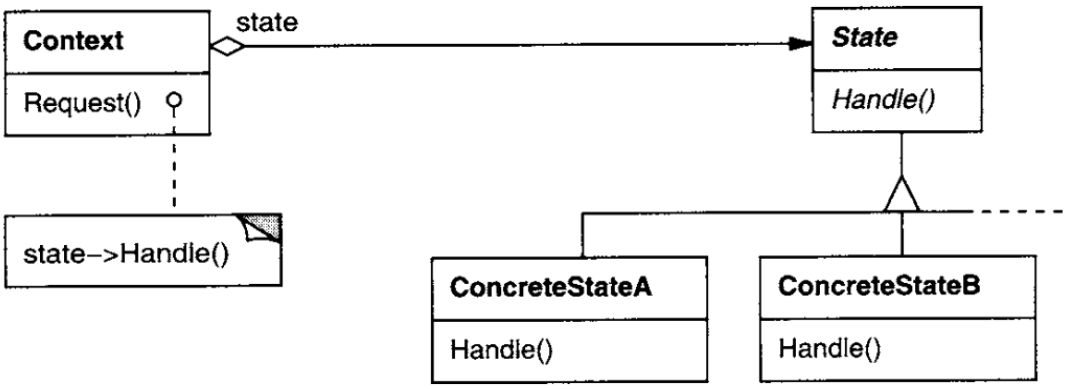
Diversos jogos de MMORPG possuem o recurso de criação de grupos para união de jogadores, chamados comumente de guildas ou clãs. Em muitos deles, como o caso de *Black Desert Online SA*, quando os jogadores são bem-sucedidos ao completar uma missão, a guilda ganha pontos de experiência. Assim como acontece com os personagens, alcançado um dado valor de experiência, a guilda evolui de nível, ganhando pontos que são usados para a aquisição de habilidades para a guilda. Essas habilidades são na verdade concedidas aos jogadores que participam da guilda. Dessa maneira, identificamos a aplicabilidade do padrão de projetos *Observer*. A guilda é o alvo a ser observado (*subject*) e os personagens que nela estão são os observadores (*observers*), de maneira que a guilda ao conseguir

uma nova habilidade, os personagens dela são notificados dessa nova habilidade e atualizam sua própria lista de habilidades.

4.3.8 *State*

O padrão de projetos *State* permite a um objeto mudar de comportamento como se tivesse mudando de classe, sem de fato fazê-lo, através da mudança de estado interno (GAMMA et al., 2007).

Diagrama 4-41 – Estrutura base do padrão de projetos *State*.



Fonte: Erich Gamma (et al., 2007)

Para o padrão de projetos *State* os trabalhos levantados pelo mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), atribuem efeitos positivos nas seguintes propriedades:

- Acoplamento
- Coesão
- Compreensibilidade
- Manutenibilidade
- Polimorfismo

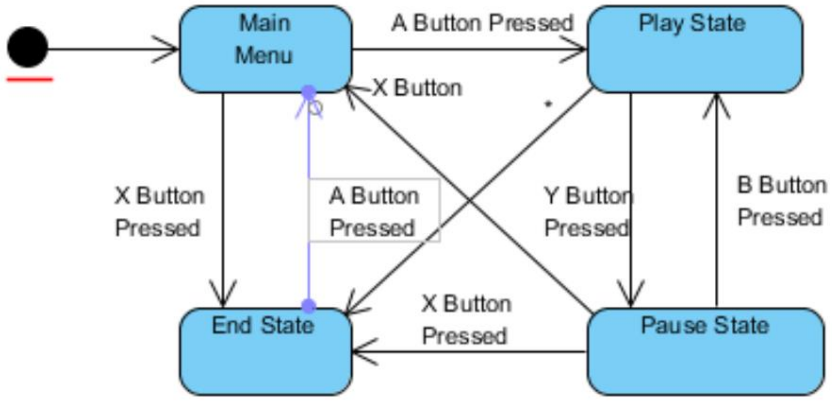
Há dois casos para a aplicabilidade do padrão de projetos *State* (GAMMA et al., 2007):

- Um objeto possui um estado que pode mudar em tempo de execução, e, há comportamentos do objeto que mudam conforme o estado muda;
- Há operações complexas que dependente do estado do objeto e possuem várias alternativas.

4.3.1.15. Uso em Jogos Digitais

Em vários jogos digitais, principalmente os de *Single Player* (KLEVJER, 2006), há a opção de pausar o jogo para o uso de um menu (salvar ou carregar um estado do jogo, sair, alterar configurações gráficas etc.). Nesse momento, na maioria dos jogos o jogo em si é paralisado, e, quando o jogador sai do menu, o jogo continua de onde foi paralisado.

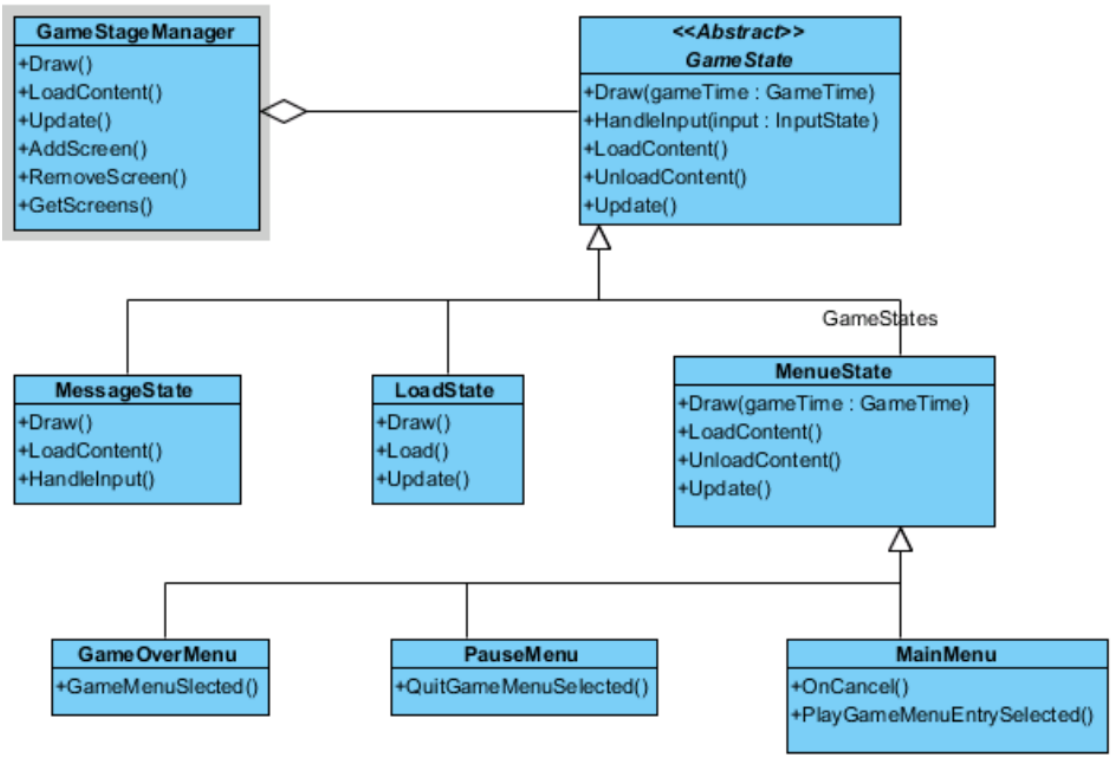
Diagrama 4-42 – Estados de jogo sem o padrão *State*.



Fonte: Junfeng Qu, Yinglei Song e Yong Wei (2013). Demonstração da estrutura de estados de jogos em que é possível pausá-los.

Comumente os jogos que atendem a esse escopo possuem os estados “jogando”, “pausado” e “finalizado”, representados respectivamente no Diagrama 4-49 como “*Play State*”, “*Pause State*” e “*End State*”.

Diagrama 4-43 – Estados de jogo com o padrão *State*.



Fonte: Junfeng Qu, Yinglei Song e Yong Wei (2013). Implementação do padrão de projetos State para abordar a mudança de estado do jogo, como *Pause* e *Game Over*.

Nessa função, o autor Junfeng propõe o uso do padrão de projetos *State* para o manuseio dos estados de “exibindo mensagem”, “carregando jogo”, “acessando menu de jogo acabado”, “acessando menu de pausa” e “acessando menu principal”. O Diagrama 4-50 demonstra que a classe *GameStageManager* é quem mantém o estado concreto do jogo, que pode variar entre os estados *MessageState*, *LoadState* e *MenueState* sem grande complexidade.

4.3.1.16. Equivalência com Jogo Digital Moderno

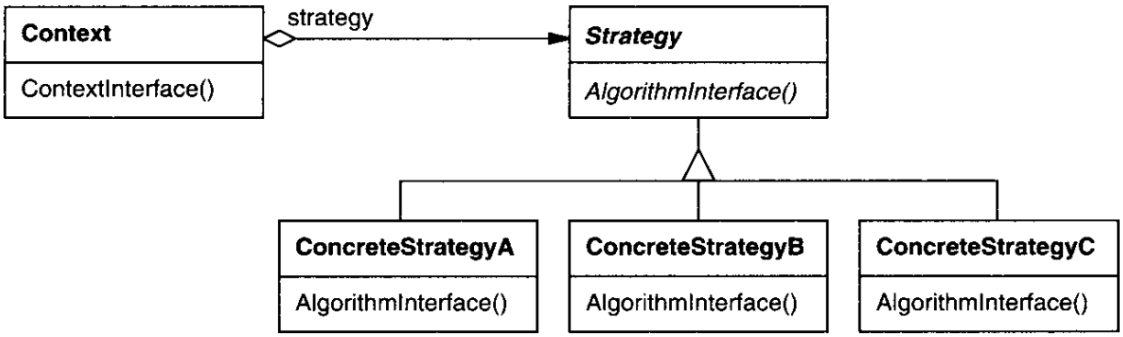
O jogo *Pro Evolution Soccer 2018* possui, tanto no modo *Single Player* quanto *Multiplayer* (KLEVJER, 2006), o recurso de pausa do jogo (durante a partida), salvar o estado (campeonatos, modo carreira, treinamento), carregar um estado salvo (os mesmos de salvar estado), partida em andamento, menu de pausa etc. Todos os estados podem ser implementados com o padrão de projetos *State*.

Partindo de uma abordagem mais fundamentalista, autor Ewerton Menezes (2012) identifica a presenta de estados em todos os jogos. O sentido do termo “estado” para Ewerton é ligeiramente menos profundo que o aplicado por Junfeng Qu, Yinglei Song e Yong Wei, mas também partilham das duas aplicabilidades do padrão *State* descritas por Gamma et al.

4.3.9 *Strategy*

O padrão de projetos *Strategy* intenciona definir uma família de algoritmos que são encapsulados e trocados entre si em tempo de execução (GAMMA et al., 2007). Dessa maneira, o padrão de projetos *Strategy* permitir que objetos mudem de comportamento em tempo de execução de forma ilimitada.

Diagrama 4-44 – Estrutura base do padrão de projetos *Strategy*.



Fonte: Erich Gamma (et al., 2007)

Os autores Ampatzoglou, Charalampidou e Stamelos (2013), traz um mapeamento de trabalhos que apontam efeitos positivos do padrão de projetos *Strategy* nas seguintes propriedades:

- Compreensibilidade
- Manutenibilidade
- Polimorfismo

Para o padrão de projetos *Strategy*, o autor Erich Gamma (et al., 2007) atribui as seguintes aplicabilidades:

- Há muitas classes relacionadas que diferem apenas em seu comportamento;
- É preciso criar variantes de algoritmos;
- É desejável evitar que estruturas de dados complexas sejam expostas;

- Uma classe possui um comportamento complexo que varia confirme uma condição e é desejável segregar as alternativas resultantes da condição.

4.3.1.17. Uso em Jogos Digitais

No jogo digital de cartas criado por Hansen (2004), há uma variação de cartas através de quatro atributos:

- Cor: Cada carta pode ser roxa, verde ou vermelha.
- Número: Cada carta possui de um a três símbolos.
- Símbolo: Cada símbolo pode ser um círculo oval, um diamante ou rabisco.
- Sombreamento: Os símbolos de uma mesma carta podem ser sólidos, abertos ou listrados.

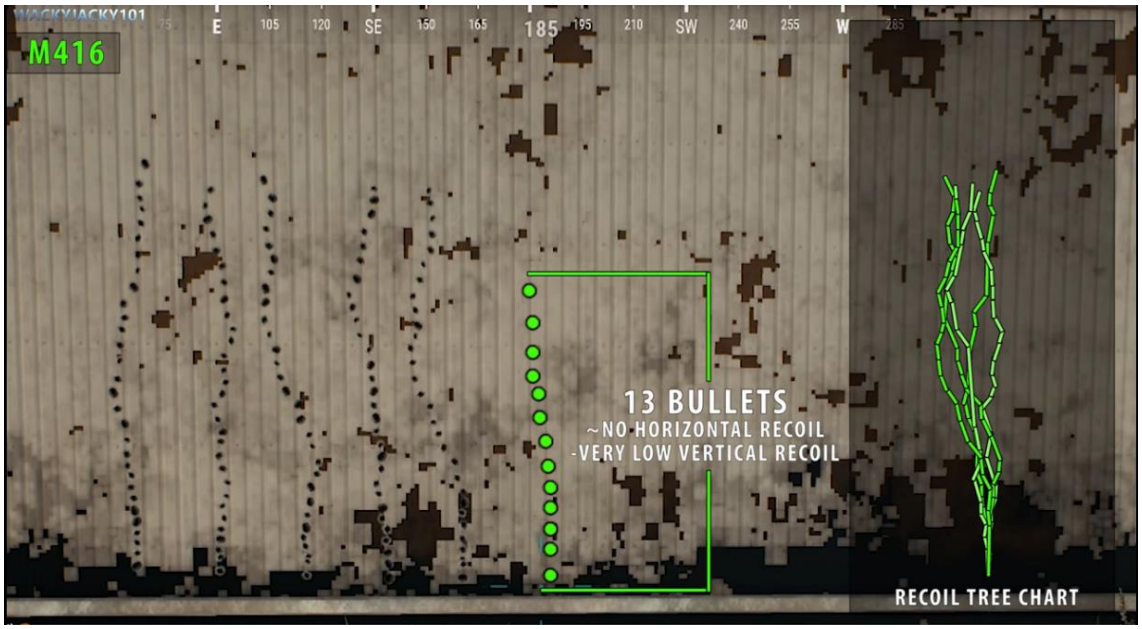
O deck de cartas é feito com uma carta para cada combinação possível. Assim, dada a fórmula (cores*número*símbolos*sombreamentos = 3*3*3*3) temos um deck com 81 cartas únicas.

Sabendo que esse valor de combinações torna o algoritmo de criação complexo de se manter, Hansen propõe o uso do padrão de projetos *Strategy* para criação das variantes símbolo e sombreamento, encapsulando os comportamentos de desenhar símbolos e sombreamentos das cartas, criando uma família de comportamentos para símbolos e outra para sombreamentos, sendo possível defini-los em tempo de execução e possibilitando uma independência entre eles e uma melhor manutenibilidade.

4.3.1.18. Equivalência com Jogo Digital Moderno

O padrão de projetos *Strategy* possui um poder imenso no universo dos jogos digitais. Uma equivalência simples e direta com a proposta dada por Hansen, é o recuo – popularmente conhecido como “coice”) das armas no jogo *Player’s Unknows Battlegrounds*. Ao atirar em sequência, o recuo resulta no que se chama *spray* da arma, que é o desenho irregular feito pelas balas da arma.

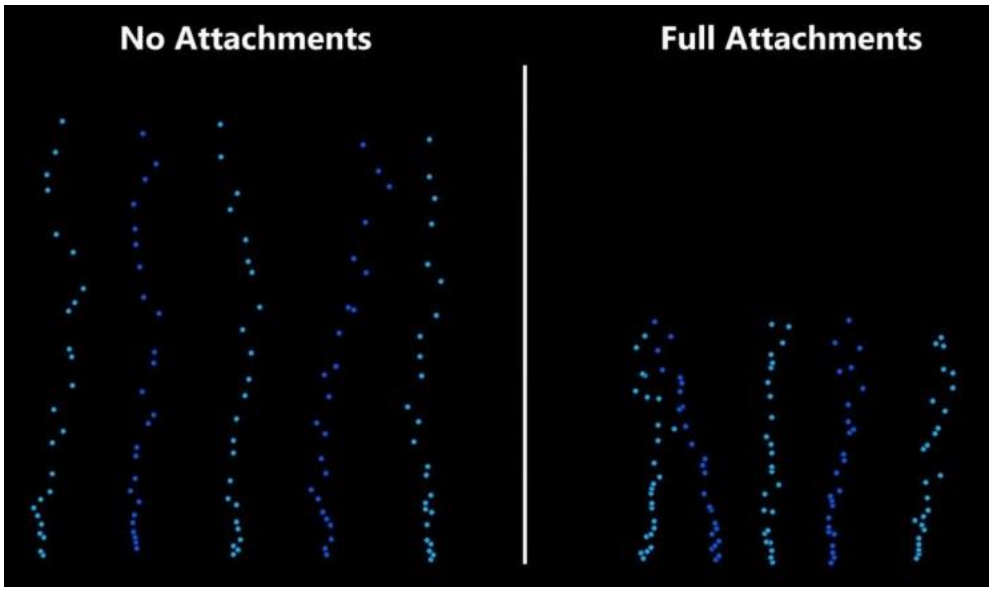
Figura 4-8 – Sprays de M416 em *Player's Unknows Battlegrounds*.



Fonte: Wacky Jacky 101 (2018). Demonstração de seis sprays dados pela arma M416 do jogo *Player's Unknows Battlegrounds*.

Das dezenas de armas de fogo que o jogo dispõe, cada uma delas têm um recuo diferente ao atirar, que sofre alterações individuais nas três posições possíveis ao personagem, que são: de pé, agachado e deitado; e também algumas armas têm o recuo alterado pelos acessórios que nela são anexados no intuito de facilitar o controle desse recuo para o jogador. A Figura 4-9 elucida o comportamento de recuo.

Figura 4-9 – Sprays de Vector em *Player's Unknows Battlegrounds*.



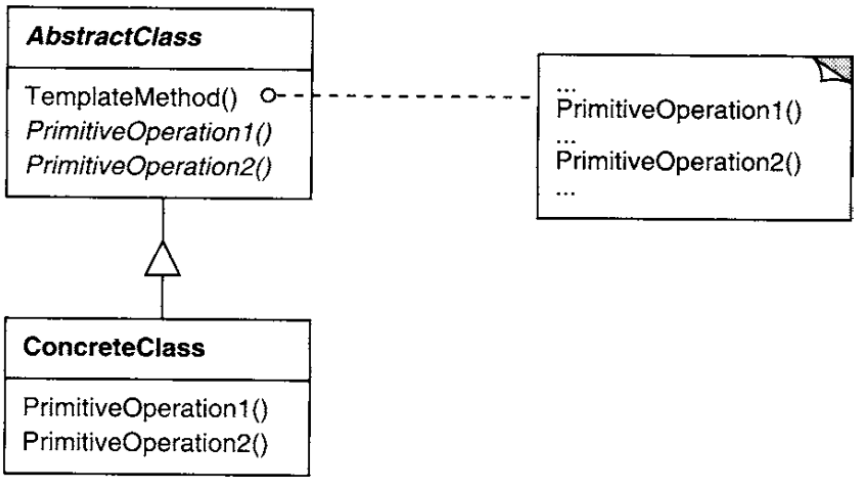
Fonte: Wacky Jacky 101 (2018). Demonstração do spray da arma "*Vector*" do jogo *Player's Unknows Battlegrounds*. A esquerda o spray com a arma sem acessórios. A direita, a arma com todos os melhores acessórios possíveis, que concedem maior precisão ao jogador atirar, devido à redução do recuo da arma.

É contundente que atirar é um comportamento de todo personagem com arma de fogo. E esse comportamento é alterado conforme a arma utilizada (que individualmente tem sua complexidade ao usar acessórios, complexidade essa que possui sua devida proposta de solução na descrição do padrão de projetos Decorator). Assim, com cada arma possuindo seu comportamento de tiro encapsulado, o comportamento de atirar do personagem, é atribuído em tempo de execução ao jogador equipar a arma.

4.3.10 *Template Method*

Segundo Gamma (et al., 2007), o padrão de projetos *Template Method* intenciona criar uma abstração de estrutura base para operações, delegando a implementação e a especificidade individual para as subclasses da abstração, mantendo a estrutura base.

Diagrama 4-45 – Estrutura base do padrão de projetos *Template Method*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Template Method* nas propriedades:

- Adaptabilidade
- Estabilidade
- Manutenibilidade

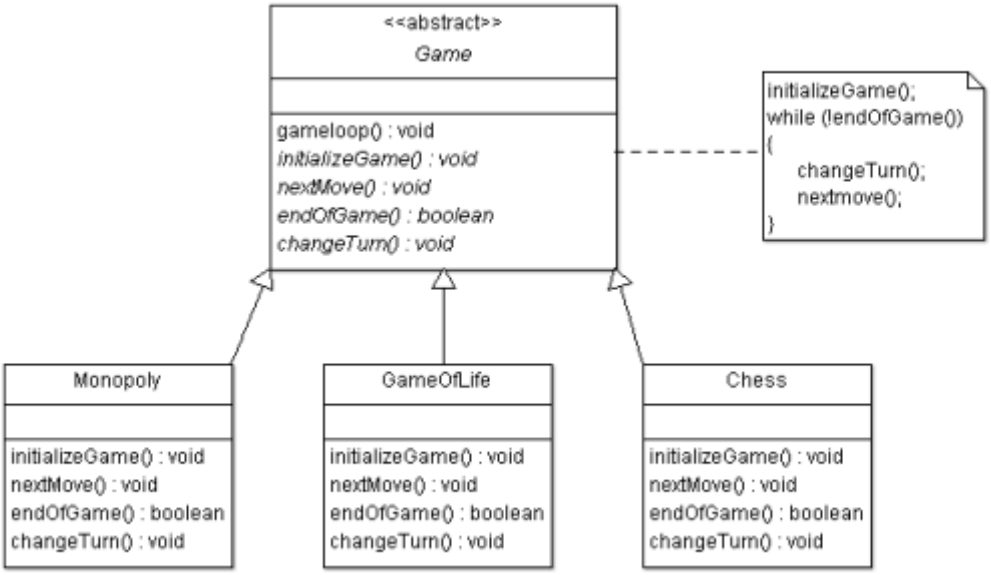
O padrão de projetos *Template Method* possui as seguintes aplicabilidades (GAMMA, et al., 2007):

- É desejável implementar as partes do algoritmo que não variam, deixando que as subclasses implementem as especificidades, evitando replicação de códigos.

4.3.1.19. Uso em Jogos Digitais

Muitos jogos compartilham de uma mesma mecânica, principalmente quando fazem parte do mesmo gênero. Para jogos baseados em turnos, Ampatzoglou, Kounoukla e Anagnostopoulos (2016) propõem um motor de criação de jogos desse gênero utilizando o padrão de projetos *Template Method*.

Diagrama 4-46 – Motor de turnos com o padrão *Template Method*.



Fonte: Ampatzoglou, Kounoukla e Anagnostopoulos (2016). Proposta para motor de jogos baseados em turnos, utilizando o padrão de projetos *Template Method*.

Como representado no Diagrama 4-55, a classe abstrata *Game* utiliza o método *gameloop()* para estruturar a ordem de chamada aos métodos *initializeGame(), nextMove(), endOfGame()* e *changeTurn()*. Essa ordem é compartilhada por todos os jogos criados nesse motor, onde cada jogo especificará o algoritmo contido nos métodos citados em *gameloop()*.

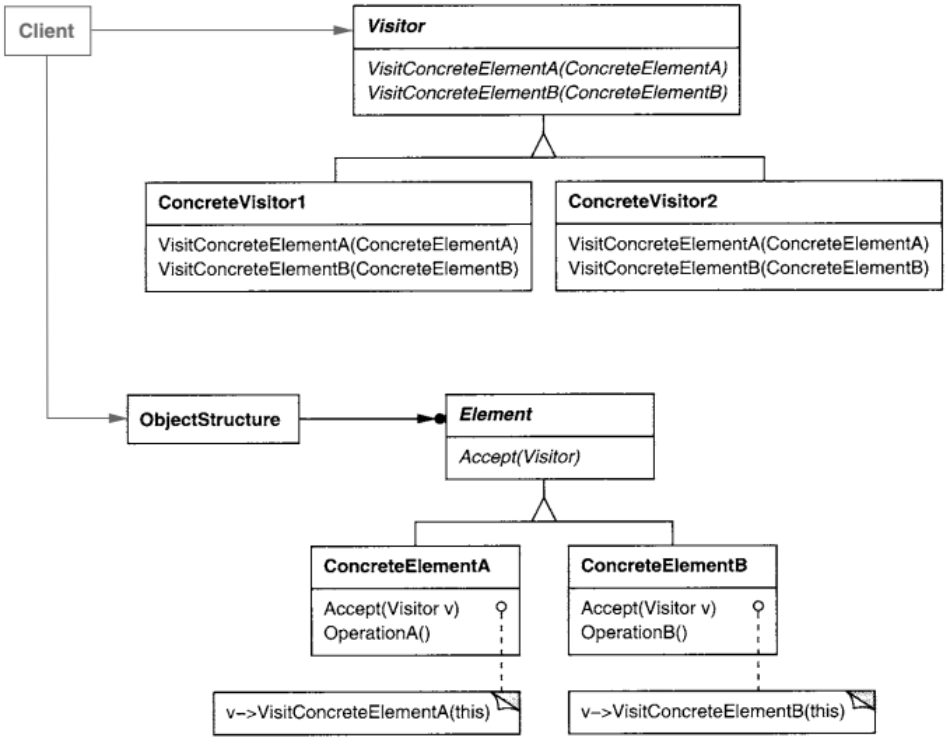
4.3.1.20. Equivalência com Jogo Digital Moderno

A proposta dada por Ampatzoglou, Kounoukla e Anagnostopoulos (2016), traz em si, por consequente do contexto genérico, a aplicabilidade do *Template Method* em um motor de jogos para não só o gênero de jogos baseados em turnos, mas também qualquer jogo que possua uma mecânica que possa ser padronizada em partes através de um *gameloop*. O autor Nystrom (2014) afirma que a maioria dos jogos possui um *gameloop* que é uma estrutura executada continuamente enquanto o jogo funciona, e, em cada ciclo do *loop*, vários algoritmos são executados como animações, renderização, verificação de colisões, inteligência artificial etc. A descrição de Nystrom reforça a proposta de Ampatzoglou, Kounoukla e Anagnostopoulos, e confirma a acentuada abrangência do *gameloop* e por consequente o uso do padrão de projetos *Template Method*.

4.3.11 *Visitor*

O padrão *Visitor* representa uma operação realizada sobre elementos da estrutura de um objeto. Ele permite criar uma nova operação sem que se mude a classe já estabelecida dos elementos sobre as quais ela opera (GAMMA et al., 2007). Dessa maneira, separa-se o algoritmo em si da estrutura do objeto. Literalmente o algoritmo de uma função é desacoplado da classe que comumente o faria, para que, com cada função em uma nova e própria classe, essa classe que era detentora dessas funções passe agora a se permitir “visitar” por “visitadores”; os visitadores são as novas classes que possuem as funções que foram desacopladas.

Diagrama 4-47 – Estrutura base do padrão de projetos *Visitor*.



Fonte: Erich Gamma (et al., 2007)

O mapeamento de Ampatzoglou, Charalampidou e Stamelos (2013), cita trabalhos que apontam efeitos positivos do padrão de projetos *Visitor* nas propriedades:

- Acoplamento
- Herança

- Manutenibilidade

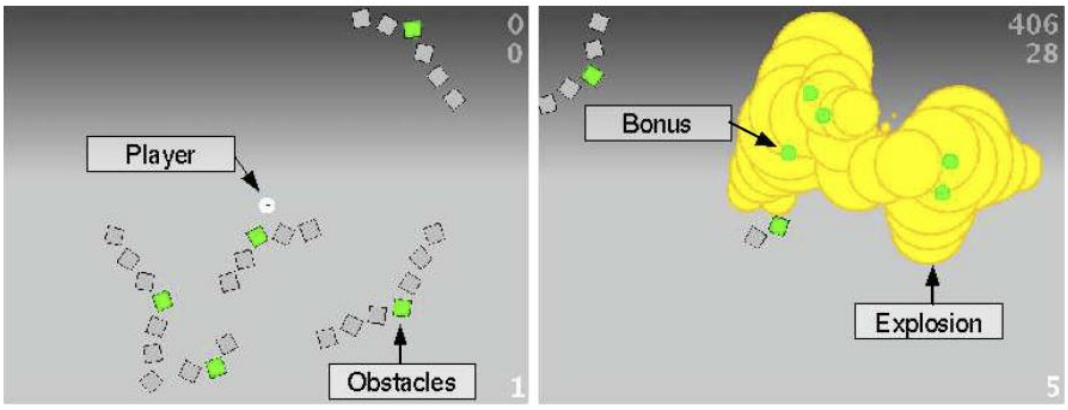
Segundo Gamma (et al., 2007) o padrão de projetos *Visitor* possui as seguintes aplicabilidades:

- Em uma estrutura de objetos que possui muitas classes de objetos, porém, com diferentes interfaces, e que para executar operações nesses objetos são necessárias as suas classes concretas;
- Evitar poluição de classes que necessitam de muitas operações.
- Ainda que a estrutura de objetos dificilmente mude, é comum que sejam criadas novas operações e isso requer uma redefinição da interface para todos os visitantes.

4.3.1.21. Uso em Jogos Digitais

O trabalho sobre ensinar padrões de projetos através da aplicação em jogos digitais, de Gestwicki e Sun (2008), traz o jogo *EEClone*. Nele o jogador controla uma bomba em um mundo cheio de obstáculos voadores e bônus também voadores, mostrado na Figura 4-10. Se o jogador colidir com um obstáculo, ele perde uma vida. Se ele detonar a bomba também perde uma vida, entretanto, essa detonação também pode iniciar uma cadeia de explosões uma vez que explodir a bomba próximo de um bônus faz com que eles explodam também, e, bônus explodidos geram pontos que por sua vez concedem vidas.

Figura 4-10 – Comportamentos do jogo *EEClone*.

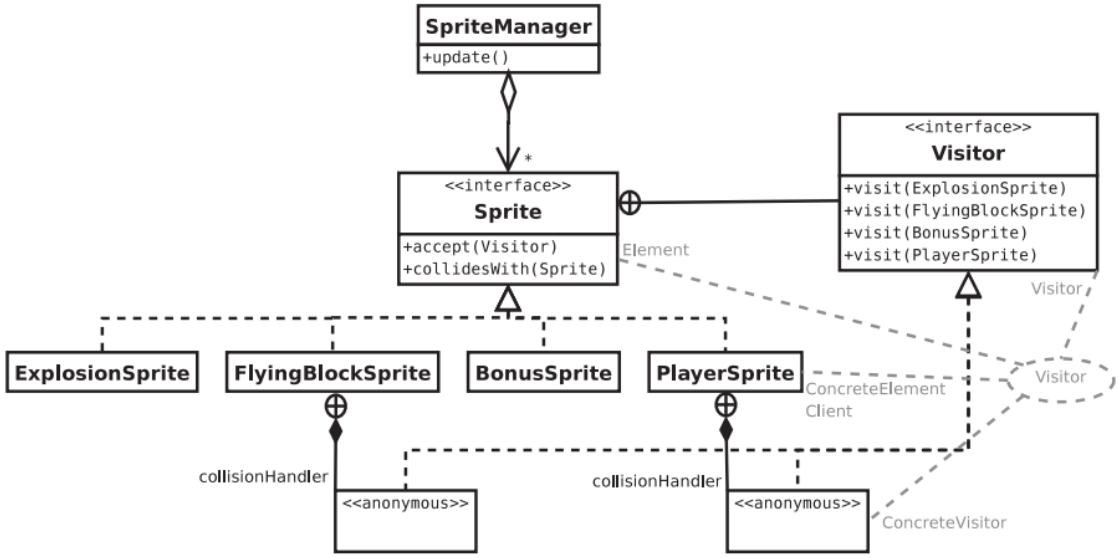


Fonte: Gestwicki e Sun (2008). Na esquerda o personagem com a bomba, desviando de obstáculos (cinzas) e tentando colidir com os bônus (verdes). Na direita ele tem detonado a bomba,

que acarretou numa série de explosões entre os bônus que estavam próximos numa reação em cadeia.

Como demonstrado na Figura 4-11, o *player* e os obstáculos são *visitors*. O *PlayerSprite* contém um algoritmo para a colisão com *BonusSprite*, que gera pontos; e para a colisão com *FlyingBlockSprite* (obstáculo), que remove uma vida do jogador. Já o algoritmo do *visitor FlyingBlockSprite* está para a colisão com *ExplosionSprite*, que fará o *FlyingBlockSprite* explodir também e gerar mais pontos para o jogador. Os demais métodos os quais a colisão é nula, ficam vazios. Dessa maneira, havendo qualquer mudança nas colisões, inclusive nas que estão nulas, o desacoplamento dos algoritmos permite uma alteração clara e sem refatorações.

Figura 4-11 – Uso do padrão *Visitor* no *EEClone*.



Fonte: Gestwicki e Sun (2008). Estrutura de classes do jogo *EEClone* implementadas com o padrão *Visitor*. Os obstáculos representados por *FlyingBlockSprite*, o bônus por *BonusSprite*, a explosão por *ExplosionSprite* e o jogador pela *PlayerSprite*.

4.3.1.22. Equivalência com Jogo Digital Moderno

O esquema de colisão desenvolvido por Gestwicki e Sun pode facilmente ser adaptado às colisões presentes na maioria dos jogos modernos. O jogo *Player's Unknows Battlegrounds* possui uma série de diferentes comportamentos para colisões, como explosões de granadas que causam danos a jogadores e veículos, mas em objetos sólidos do cenário apenas deixam a marca de explosão; o impacto de balas em jogadores e veículos causando danos, em janelas as destruindo e em

objetos sólidos como paredes, deixando buracos visíveis etc. A lista de colisões é grande e o uso do padrão *Visitor* nesse contexto é de grande impacto na manutenibilidade e estabilidade.

4.4 ANÁLISE DAS EVIDÊNCIAS

Ao término da pesquisa, através do estudo dos resultados e desenvolvimento das propostas, foi possível responder às perguntas de pesquisa:

- Quais padrões de projeto GoF são utilizados em propostas de soluções para funcionalidades de jogos digitais?

  Todos os padrões apresentaram ao menos uma proposta, ainda que alguns de proposta única, como *Chain of Responsibility* e *Interpreter*, tenham suas explicações não muito esclarecidas ou passiveis de questionamentos sobre os critérios de aplicabilidades definidos por Erich Gamma (et al., 2007).

- Há equivalência entre as funções solucionadas na literatura e funções de jogos digitais modernos?

  Com exceção do padrão *Interpreter*, cuja proposta apresentou grandes controvérsias quanto aos critérios de aplicabilidades, para todos os demais 22 padrões *GoF* foi possível propor equivalências entre funcionalidades presentes na literatura e funcionalidades de jogos digitais modernos. Ainda sobre o padrão *Interpreter*, foi possível discutir e propor uma implementação para com base no código fonte do L2J.

- Quais padrões de projetos GoF possuem avaliações de atributos de qualidade e métricas de software quando aplicados em jogos digitais?

  Para os padrões *Flyweight* e *Memento*, a soma de trabalhos positivos e negativos apresentou um resultado igual ou inferior a zero, não sendo possível identificar atributos ou métricas de qualidade de software para eles; ainda que analisando a proposta de Nystrom (2014) sobre gerador de árvores com o padrão *Flyweight* possua uma argumentação muito contundente para o atributo Reusabilidade e a métrica Tamanho.

- Os padrões de projetos GoF são relevantes no desenvolvimento de jogos digitais modernos?

Com as perguntas específicas devidamente respondidas é possível responder à pergunta principal com sólida segurança. Os padrões de projetos definidos por Erich Gamma (et al., 2007) são relevantes ao desenvolvimento de jogos modernos, uma vez que a literatura demonstra soluções de funcionalidades de jogos digitais implementadas com esses padrões, e, essas funcionalidades são mecanicamente equivalentes à várias funcionalidades de jogos digitais modernos, além de 21 dos 23 padrões GoF apresentarem, na literatura, resultados positivos quanto a avaliação de atributos e métricas de qualidade de software.

5 CONCLUSÃO

Este trabalho evidencia, através de uma revisão sistemática da literatura, a alta complexidade no desenvolvimento de jogos digitais, mesmo aos jogos de décadas atrás, e, aborda a pertinência do uso e consequente estudo dos padrões de projetos descritos por Erich Gamma (et al, 2007).

O trabalho mostra várias implementações de padrões de projetos para solucionar funcionalidades de jogos digitais, e, ainda assim, expõe o quanto esse tema ainda pode ser explorado e estendido.

Na pesquisa por atributos e métricas de software dos padrões de projetos aplicados no contexto dos jogos digitais e suas especificidades, não foram encontrados trabalhos tratando diretamente esse tema. Entretanto, foi possível apontar trabalhos que estudaram os atributos e métricas de software de padrões de projetos em contextos gerais, sem a especificidade dos jogos, mas ainda assim, trazendo informações relevantes a esse trabalho.

Para os padrões de projetos GoF, 22 dos 23 se mostraram pertinentes não só aos jogos digitais do passado, como também para os jogos digitais modernos, onde foi possível propor uma equivalência entre a funcionalidade solucionada com a aplicação do padrão de projetos descrita na literatura, e, uma funcionalidade de jogo digital moderno, de forma direta. O único padrão de projetos GoF onde, na literatura, a aplicabilidade não atendeu aos critérios, foi o *Interpreter*. Apesar disso, para o padrão de projetos *Interpreter* foi proposta uma solução para uma funcionalidade de jogos digital, abrindo margem para a possibilidade da relevância deste padrão no desenvolvimento de jogos digitais modernos.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

O trabalho abre espaço para vários outros questionamentos, como a abordagem de diversos contextos específicos, segregados por exemplo pelo gênero dos jogos digitais, como estratégia em tempo real ou tiro em primeira pessoa; ou, pela área técnica envolvida na programação de jogos digitais, como inteligência artificial ou conexão cliente-servidores.

Há grande lacuna na análise de atributos e métricas de qualidade de software dos padrões de projetos GoF no contexto dos jogos digitais, uma vez que o único

trabalho encontrando tratando esse tema foi desconsiderado por ausência de referências e argumentações em suas afirmações.

A pesquisa identificou que alguns padrões GoF têm alto quantitativo de uso em jogos digitais, enquanto outros aparecem pouco e alguns com frágil validação dos critérios aplicabilidade. Este trabalho não analisou as razões dessa observação.

Além dos padrões de projeto GoF, há outros padrões de projetos que não foram abordados nesse trabalho, como os GRASP (LARMAN, 2004), e podem ser submetidos à mesma metodologia tratada neste trabalho. Também, Nystrom (2014) traz 13 padrões construídos pela necessidade comum ao contexto dos jogos digitais, os quais também não foram abordados nesse trabalho.

Outro desdobramento possível deste trabalho seria a implementação, via código executável, para validação em última instância das propostas dadas tanto pela literatura aqui extraída e descrita, quanto das equivalências construídas nesse trabalho. Essa linha de pesquisa também alerta para a implementação dos códigos das propostas em motores de jogos, como *Unreal Engine* e *Unity3D*, analisando a aplicabilidade dos padrões de projeto para a criação de jogos digitais nesses motores, afim de encontrar possíveis considerações como a necessidade de adaptações, limitações ou facilidades.

Também, é possível que uma análise dos padrões de projetos no ambiente do mercado de trabalho com os desenvolvedores e não através da literatura científica, apresente resultados mais ricos ou mesmo divergências a serem estudadas.

REFERÊNCIAS

AMPATZOGLOU, A. e STAMELOS, I. Software engineering research for computer games: A systematic review. Information and SoftwareTechnology, 52, 9, 888–901, Setembro, 2010.

ARK: Survival Evolved. Kirkyland: Studio Wildcard, 2014. Disponível em: <http://store.steampowered.com/app/346110/ARK_Survival_Evolved>. Acesso em: 17/04/2018. Acesso em: 11/04/2018.

ARKSEY, H. e O'MALLEY, L. Scoping studies: towards a methodological framework. **International Journal of Social Research Methodology**, 8, 1, 19-32, Fevereiro, 2005.

BARTON, Matt. & LOGUIDICE, Bill. The History of Spacewar!: The Best Waste of Time in the History of the Universe. São Francisco: Gamesutra. Disponível em: <http://www.gamasutra.com/view/feature/4047/the_history_of_spacewar_the_best_.php?print=1>. Acesso em: 07/02/2017.

BASILI, Victor., BRIAND, Lionel. & MELO, Walcélio. A Validation of Object Oriented Design Metrics as Quality Indicators. **IEEE Transactions Software Engineering**, 22, 10, 751-761, Outubro, 1996.

Bitesize: Hexadecimal and character sets. BBC, 2018. Disponível em: <https://www.bbc.com/education/guides/zp73wmn/revision>. Acesso em 28/04/2018.
Black Desert Online SA. Anyang: Pearl Abyss, 2010. Disponível em: <https://store.steampowered.com/app/706220/Black_Desert_Online_SA>. Acesso em: 11/04/2018.

BUCANEK, James. **Learn Objective-C for Java Developers**. New York: Apress, 2009.

CASTRO, A. Revisão sistemática e Meta-análise. São Paulo: UNIFESP, 2001. Disponivel em: <https://pt.slideshare.net/ptsuneda/reviso-sistemtica-e-meta-anlise>. Acesso em: 29/01/2018.

COUTINHO, Dário. Street of Rage: curiosidades e polemicas sobre o clássico do Mega Drive. TECHTUDO, 2014. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2014/10/streets-rage-curiosidades-e-polemicas-sobre-o-classico-do-mega-drive.html>. Acesso em 28/04/2018.

DOMHOLD, E. **Rehabilitation Research: Principles and Applications**. Saint Louis: Elsevier Saunders, 2005.

F1™ 2017. Warwickshire: Codemasters, 1986. Disponível em: <http://store.steampowered.com/app/515220/F1_2017>. Acesso em: 17/04/2018.

Fall of Light. Placência: Runeheads, 2017. Disponível em: <https://store.steampowered.com/app/633950/Fall_of_Light>. Acesso em: 17/04/2018.

Foxhole. Toronto: Calopfoot, 2017. Disponível em: <https://store.steampowered.com/app/505460/Foxhole>. Acesso em: 11/04/2018.

HUIZINGA, J. **Homo Ludens (1938)**. São Paulo: Editora Perspectiva, 2000.

ISO/IEC 25010: 2011. **Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models**.

JAVA SE documentation: Interface Iterator. ORACLE, 2004. Disponível em: <https://docs.oracle.com/javase/1.5.0/docs/api/java/util/Iterator.html>. Acesso em 28/04/2018. Acesso em 28/04/2018.

KHOMH, Foutse. & GUÉHÉNEUC, Yann-Gaël. Do design patterns impact software quality positively?. **12th European Conference on Software Maintenance and Reengineering**, 274–278, Abril, 2008.

KITCHENHAM, B. et al. Guidelines for performing Systematic Literature Reviews in Software Engineering Version 2.3. Keele: Keele University e University of Durham, 2007. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.493.910&rep=rep1&type=pdf>. Acesso em: 11/06/2018.

KLEVJER, Rune. What is the Avatar? Fiction and Embodiment in Avatar-Based Singleplayer Computer Games. 2006. Dissertation for the degree doctor rerum politicarum – Department of Information Science and Media Studies, University of Bergen, Bergen, 2006.

LARMAN, Craig. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development** – 2ª ed. Upper Saddle River: Prentice Hall, 2004.

League of Angels II. Honk Kong: YOUZU GAMES HONGKONG LIMITED, 2009. Disponível em: <http://loa2.gtarcade.com/>. Acesso em: 17/04/2018.

LONQUIST, J. Condado de Santa Clara: Computer History Museum, 1996. Disponível em: <http://www.computerhistory.org/revolution/computer-games/16/189/2213>. Acesso em: 17/04/2018.

MCDONALD, Emma. The global games market will reach $108.9 billion in 2017 with Mobile taking 42%. São Francisco: Newzoo. Disponível em: <https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42>. Acesso em: 01/05/2017.

MOGNON, Mateus. PUBG Ultrapassa Minecraft e se torna o game mais vendido na história do PC. Florianópolis: Adrenaline LTDA, 2018. Disponível em: <https://adrenaline.uol.com.br/2018/01/10/53872/pubg-ultrapassa-minecraft-e-se-torna-o-game-mais-vendido-na-historia-do-pc>. Acesso em 28/04/2018.

MORANDINI, Andrea. Como jogar xadrez: regras e noções básicas. Chess.com, 2017. Disponível em: <https://www.chess.com/pt-BR/learn-how-to-play-chess>. Acesso em 19/04/2018.

NCSOFT. Lineage 2. Disponível em: <lineage2.com>. Acesso em: 17/04/2018.
PERUCIA, A. S. et al. **Desenvolvimento de Jogos Eletrônicos – Teoria e Prática**. São Paulo: Novatec, 2005.

Player's Unknows Battlegrounds. Seoul: PUBG CORPORATION, 2017. Disponível em: <https://store.steampowered.com/app/578080/PLAYERUNKNOWNS_BATTLEGROUNDS/>. Acesso em: 11/04/2018.

PRESSMAN, Roger. **Engenharia de Software** – 7ª ed. Porto Alegre: AMGH Editora, 2011.
Pro Evolution Soccer, 2017. Tóquio: Konami Corporation, 1969. Disponível em: <http://store.steampowered.com/app/592580/PRO_EVOLUTION_SOCCER_2018>. Acesso em: 17/04/2018.

SCHELL, J. **A Arte de Game Design**: o livro original. São Paulo: Editora Campus, 2010.

SCHREIBER, I. Game Design Concepts: An experiment in game design and teaching. gamedesignconcepts.wordpress.com, 2009. Disponível em: <http://gamedesignconcepts.wordpress.com/2009/06/29/level-1-overview-what-is-a-game/>. Acesso em 28/05/2018.

SOMMERVILLE, Ian. **Engenharia de Software** – 9ª ed. São Paulo: Pearson Prentice Hall, 2011.

SpellForce 3. Munique: Grimlore Games, 2013. Disponível em: <http://store.steampowered.com/app/311290/SpellForce_3>. Acesso em: 11/04/2018.

STATISTA. Genre breakdown of video game sales in the United States in 2016. Nova York, Statista, 2017. Disponível em: <https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre>. Acesso em: 07/05/2017.

STEAM: Estatísticas do Steam e de jogos. Luxemburgo: Valve Corporation, 2018. Disponível em: <http://store.steampowered.com/stats>. Acesso em: 17/04/2018.

STEAM: Melhores de 2017: Lançamentos. Luxemburgo: Valve Corporation, 2018. Disponível em: <http://store.steampowered.com/sale/2017_top_new_releases>. Acesso em: 17/04/2018.

TORRES, Taíza. Banco imobiliário: 7 benefícios do jogo na educação. Canal do Ensino, 2012. Disponível em: <https://canaldoensino.com.br/blog/banco-imobiliario-7-beneficios-do-jogo-na-educacao>. Acesso em 19/04/2018.

TOTOV, Ivan et al. L2J Server Project. L2J Server, 2004. Disponível em: <https://bitbucket.org/l2jserver/l2j_server/overview>. Acesso em 19/04/2018.

TRAVASSOS, G. H. e BIOLCHINI, J. Revisões Sistemáticas Aplicadas a Engenharia de Software. XXI Simpósio Brasileiro de Engenharia de Software. 2007.

Ultima series. Moby Games, 1999. Disponível em: <http://www.mobygames.com/game-group/ultima-series>. Acesso em: 07/05/2017.

Unity®. Unity Tecnologies, 2018. Disponível em: <https://unity3d.com>. Acesso em 17/02/2018.

Unreal Engine 4. Epic Games, 2004. Disponível em: <https://www.unrealengine.com>. Acesso em 17/02/2018.

Unruly Heroes. Montpellier: Magic Design Studios, 2015. Disponível em: <https://store.steampowered.com/app/780350/Unruly_Heroes>. Acesso em: 17/04/2018.

Wacky Jacky 101. Youtuber Gamer. 2018. Disponível em: < https://www.youtube.com/channel/UCfK7qpS954pk14YvDCsWCuw>. Acesso em: 11/05/2018.

Wizardry games. Moby Games, 1999. Disponível em: <https://www.mobygames.com/game-group/wizardry-games>. Acesso em: 07/05/2017.

APÊNDICE A – Trabalhos incluídos

ID	Titulo	Autor	Ano
TI_01	A design goal and design pattern based approach for development of game engines for mobile platforms	PEKER, A. & CAN, T.	2011
TI_02	A Metamodeling Approach to Pattern-Based Model Refactoring	FRANCE, R. et al.	2003
TI_03	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.	2013
TI_04	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.	2016
TI_05	Design Patterns for Games	NGUYEN, D. WONG, S.	2002
TI_06	Evaluation of object-oriented design patterns in game development	AMPATZOGLOU, A. and CHATZIGEORGIOU, A.	2006
TI_07	Exploring game architecture best-practices with classic space invaders	KEENAN, E. & STEELE, A.	2011
TI_08	Game Programming Patterns	NYSTROM, R.	2014
TI_09	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.	2016
TI_10	MQP - Design Patterns in Video Game Programming	ZAWADA, L.	2013
TI_11	Object-oriented design patterns in Fortran 90/95: mazev1, mazev2 and mazev3	DECYK, V., GARDNER, H.	2008
TI_12	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.	2014
TI_13	Padrões de Projeto: Soluções reutilizaveis de software orientado a objetos	GAMMA, E. et al.	2007
TI_14	Research state of the art on GoF design patterns: A mapping study	AMPATZOGLOU, A., CHARALAMPIDOU, S. e STAMELOS, I.	2013
TI_15	Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life	WICK, R.	2005
TI_16	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.	2008
TI_17	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.	2009
TI_18	Teaching Object-Oriented Programming Laboratory With Computer Game Programming	CHEN, Y. & CHENG, Y.	2007
TI_19	The Game of Set R – An Ideal Example for Introducing Polymorphism and Design Patterns	HANSEN, S.	2004
TI_20	Use a Cabeça - Padrões de Projetos	FREEMAN, E. et al.	2007

APÊNDICE B – Trabalhos não obtidos

Titulo	Autor	Ano
Introducing Design Patterns and Best Practices in Unity	BUCHER, N.	2017
Design patterns to improve the accessibility of video games for people with visual problems	LOPEZ, J. et al.	2016
Using a card game for teaching design patterns	PICCOLO, P., & GUERRA, E.	2016

APÊNDICE C – Trabalhos que apresentam soluções para funcionalidades de jogos digitais através do uso de padrões de projetos gof criacionais

Padrão Criacional	Título	Autor
Abstract Factory	A design goal and design pattern based approach for development of game engines for mobile platforms	PEKER, A., CAN, T.
Abstract Factory	A Metamodeling Approach to Pattern-Based Model Refactoring	FRANCE, R. et al.
Abstract Factory	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
Abstract Factory	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Abstract Factory	Padrões de Projeto: Soluções reutilizaveis de software orientado a objetos	GAMMA, E. et al.
Abstract Factory	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.
Builder	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Builder	Padrões de Projeto: Soluções reutilizaveis de software orientado a objetos	GAMMA, E. et al.
Factory Method	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.
Factory Method	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
Factory Method	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Factory Method	MQP - Design Patterns in Video Game Programming	ZAWADA, L.
Factory Method	Object-oriented design patterns in Fortran 90/95: mazev1, mazev2 and mazev3	DECYK, V., GARDNER, H.
Factory Method	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Factory Method	The Game of Set R – An Ideal Example for Introducing Polymorphism and Design Patterns	HANSEN, S.
Factory Method	Padrões de Projeto: Soluções reutilizaveis de software orientado a objetos	FIGUEIREDO, R.
Factory Method	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.
Prototype	A design goal and design pattern based approach for development of game engines for mobile platforms	PEKER, A., CAN, T.
Prototype	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
Prototype	Game Programming Patterns	NYSTROM, R.

Prototype	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Prototype	Padrões de Projeto: Soluções reutilizaveis de software orientado a objetos	GAMMA, E. et al.
Prototype	Use a Cabeça - Padrões de Projetos	FREEMAN, E. et al.
Singleton	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Singleton	Game Programming Patterns	NYSTROM, R.
Singleton	MQP - Design Patterns in Video Game Programming	ZAWADA, L.
Singleton	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Singleton	Padrões de Projeto: Soluções reutilizaveis de software orientado a objetos	GAMMA, E. et al.
Singleton	Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life	WICK, R.
Singleton	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.

APÊNDICE D – Trabalhos que apresentam soluções para funcionalidades de jogos digitais através do uso de padrões de projetos gof estruturais

Padrão Estrutural	Título	Autor
Adapter	Design Patterns for Games	NGUYEN, D. WONG, S.
Adapter	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Bridge	A Metamodeling Approach to Pattern-Based Model Refactoring	FRANCE, R. et al.
Bridge	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
Bridge	Evaluation of object-oriented design patterns in game development	AMPATZOGLOU, A. CHATZIGEORGIOU, A.
Bridge	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Composite	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Composite	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
Composite	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Composite	Teaching Object-Oriented Programming Laboratory With Computer Game Programming	CHEN, Y. and CHENG, Y.
Decorator	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Decorator	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Façade	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Façade	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.
Flyweight	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Flyweight	Game Programming Patterns	NYSTROM, R.
Flyweight	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Flyweight	The Game of Set R – An Ideal Example for Introducing Polymorphism and Design Patterns	HANSEN, S.
Proxy	A design goal and design pattern based approach for development of game engines for mobile platforms	PEKER, A., CAN, T.
Proxy	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Proxy	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.

APÊNDICE E – Trabalhos que apresentam soluções para funcionalidades de jogos digitais através do uso de padrões de projetos gof comportamentais

Padrão Comportamental	Título	Autor
Chain of Responsibility	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Command	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.
Command	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
Command	Design Patterns for Games	NGUYEN, D. and WONG, S.
Command	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Command	Game Programming Patterns	NYSTROM, R.
Command	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Command	Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life	WICK, R.
Interpreter	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Iterator	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Iterator	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Mediator	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.
Mediator	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
Mediator	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Memento	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Memento	MQP - Design Patterns in Video Game Programming	ZAWADA, L.
Memento	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Memento	Use a Cabeça - Padrões de Projetos	FREEMAN, E. et al.
Observer	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.
Observer	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Observer	Evaluation of object-oriented design patterns in game development	AMPATZOGLOU, A. CHATZIGEORGIOU, A.
Observer	Game Programming Patterns	NYSTROM, R.

Observer	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
Observer	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Observer	Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life	WICK, R.
Observer	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.
Observer	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.
State	A design goal and design pattern based approach for development of game engines for mobile platforms	PEKER, A. and CAN, T.
State	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.
State	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
State	Design Patterns for Games	NGUYEN, D. WONG, S.
State	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
State	Evaluation of object-oriented design patterns in game development	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
State	Game Programming Patterns	NYSTROM, R.
State	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
State	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
State	MQP - Design Patterns in Video Game Programming	ZAWADA, L.
State	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
State	Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life	WICK, R.
State	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.
State	Use a Cabeça - Padrões de Projetos	FREEMAN, E. et al.
Strategy	A design goal and design pattern based approach for development of game engines for mobile platforms	PEKER, A., CAN, T.
Strategy	Design patterns applied for game design patterns	QU, J., SONG, Y. and WEI, Y.
Strategy	Design Patterns for Games	NGUYEN, D. WONG, S.

Strategy	Exploring game architecture best-practices with classic space invaders	KEENAN, E. and STEELE, A.
Strategy	Evaluation of object-oriented design patterns in game development	AMPATZOGLOU, A. CHATZIGEORGIOU, A.
Strategy	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
Strategy	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
Strategy	MQP - Design Patterns in Video Game Programming	ZAWADA, L.
Strategy	Object-oriented design patterns in Fortran 90/95: mazev1, mazev2 and mazev3	DECYK, V., GARDNER, H.
Strategy	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Strategy	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.
Strategy	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.
Strategy	The Game of Set R – An Ideal Example for Introducing Polymorphism and Design Patterns	HANSEN, S.
Strategy	Use a Cabeça - Padrões de Projetos	FREEMAN, E. et al.
Strategy	Use a Cabeça - Padrões de Projetos	FREEMAN, E. et al.
Template Method	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P. el al.
Template Method	Object-oriented design patterns in Fortran 90/95: mazev1, mazev2 and mazev3	DECYK, V., GARDNER, H.
Template Method	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Template Method	Teaching design patterns using a family of games	MARTÍN, M., DÍAZ, G. and ARROYO, J.
Visitor	Applying Design Patterns in Game Programming	QU, J., SONG, Y. and WEI, Y.
Visitor	Design Patterns for Games	NGUYEN, D. WONG, S.
Visitor	Implementing Game Mechanics with GoF Design Patterns	AMPATZOGLOU, P., KOUNOUKLA, X. and ANAGNOSTOPOULOS, K.
Visitor	Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Digitais	FIGUEIREDO, R.
Visitor	Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life	WICK, R.
Visitor	Teaching Design Patterns Through Computer Game Development	GESTWICKI, P & SUN, F.